

Discovering what is new in IBM Integration Bus v9.0

Lab exercises



Contents

LAB 1	APPLICATIONS AND LIBRARIES	5
1.1	INTRODUCTION	5
1.2	APPLICATIONS AND LIBRARIES VIEW	6
1.3	CREATING A LIBRARY AND LIBRARY REFERENCES	17
1.4	THE INDEPENDENT RESOURCES VIEW.....	32
1.5	WORKING WITH EXISTING PROJECTS	39
1.6	BAR FILE EDITOR.....	46
1.7	DEPLOYING RESOURCES.....	51
1.8	SUMMARY	53
LAB 2	DFDL AND MESSAGE MODEL TOOLING.....	55
2.1	INTRODUCTION TO MESSAGE MODEL STANDARDS	55
2.2	CREATING A CSV MESSAGE MODEL.....	56
2.3	TEST THE MESSAGE MODEL	83
LAB 3	GRAPHICAL DATA MAPPER.....	95
3.1	INTRODUCTION TO MAPPING	95
3.2	PREPARATION – BUILD THE SAMPLE CONNECTION	95
3.3	USING A SELECT IN A MAP	106
3.4	USING A STORED PROCEDURE IN A MAP	139
3.5	CLEAN UP	188
3.6	APPENDIX - JDBC CONFIGURATION	189
LAB 4	DECISION SERVICE NODE.....	193
4.1	INTRODUCTION TO IBM OPERATIONAL DECISION MANAGEMENT	193
4.2	RULES IN IBM INTEGRATION BUS – DECISION SERVICE NODE	195
4.3	BUILDING THE BUSINESS RULES.....	198
4.4	BUILD THE MESSAGE FLOW	229
4.5	TEST THE DECISION SERVICE MESSAGE FLOW	251
LAB 5	SERVICE DISCOVERY FOR DATABASE AND MQ SERVICES	263
5.1	INTRODUCTION TO DATABASE AND MQ SERVICES	263
5.2	LAB PREPARATION	263
5.3	MQ SERVICES.....	283
5.4	DATABASE SERVICES.....	298
5.5	USING DISCOVERED SERVICES	314
5.6	TEST THE MANAGE EMPLOYEE APPLICATION.....	332
APPENDIX A.	NOTICES	341
APPENDIX B.	TRADEMARKS AND COPYRIGHTS	343

THIS PAGE INTENTIONALLY LEFT BLANK

Lab 1 Applications and libraries

1.1 Introduction

WebSphere® Message Broker Version 8, the previous version of IBM® Integration Bus Version 9, introduced a new approach to the organization and development of applications, known as Applications and Libraries.

This short document describes how Applications and Libraries are manifested, how they appear to the message flow developer, and how they will appear in the runtime environment, through the Integration Bus Explorer.

- Section 1.2 shows a simple example of the Applications and Libraries view, and how to create and deploy a new Application.
- Section 1.3 shows a simple example of a Library, how it relates to an Application, and how to manage Library References.
- Section 1.4 shows how to deal with projects that are not part of an application or library.
- Section 1.5 shows how to import an existing Message Broker project into a Version 9 workspace and convert it to an application or library.
- Section 1.6 shows the changes made to the bar file editor to reflect the revised Applications and Libraries implementation.

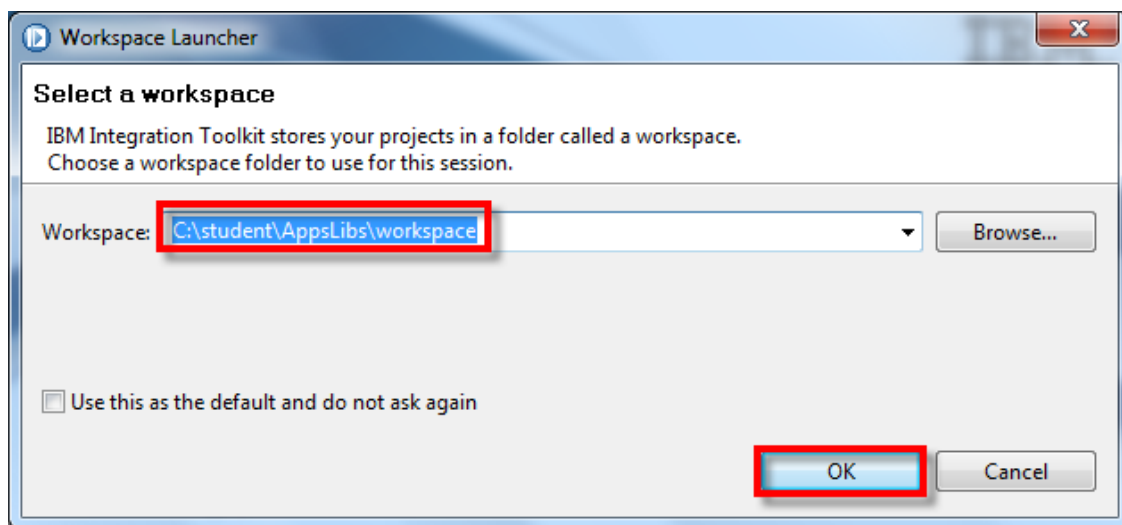
You will be able to run most of this lab with just the IBM Integration Bus Version 9 Toolkit, without the need for any additional materials.

1.2 Applications and libraries view

- __1. Click on the icon to start the IBM Integration Toolkit 9.0 (or use the Start menu).

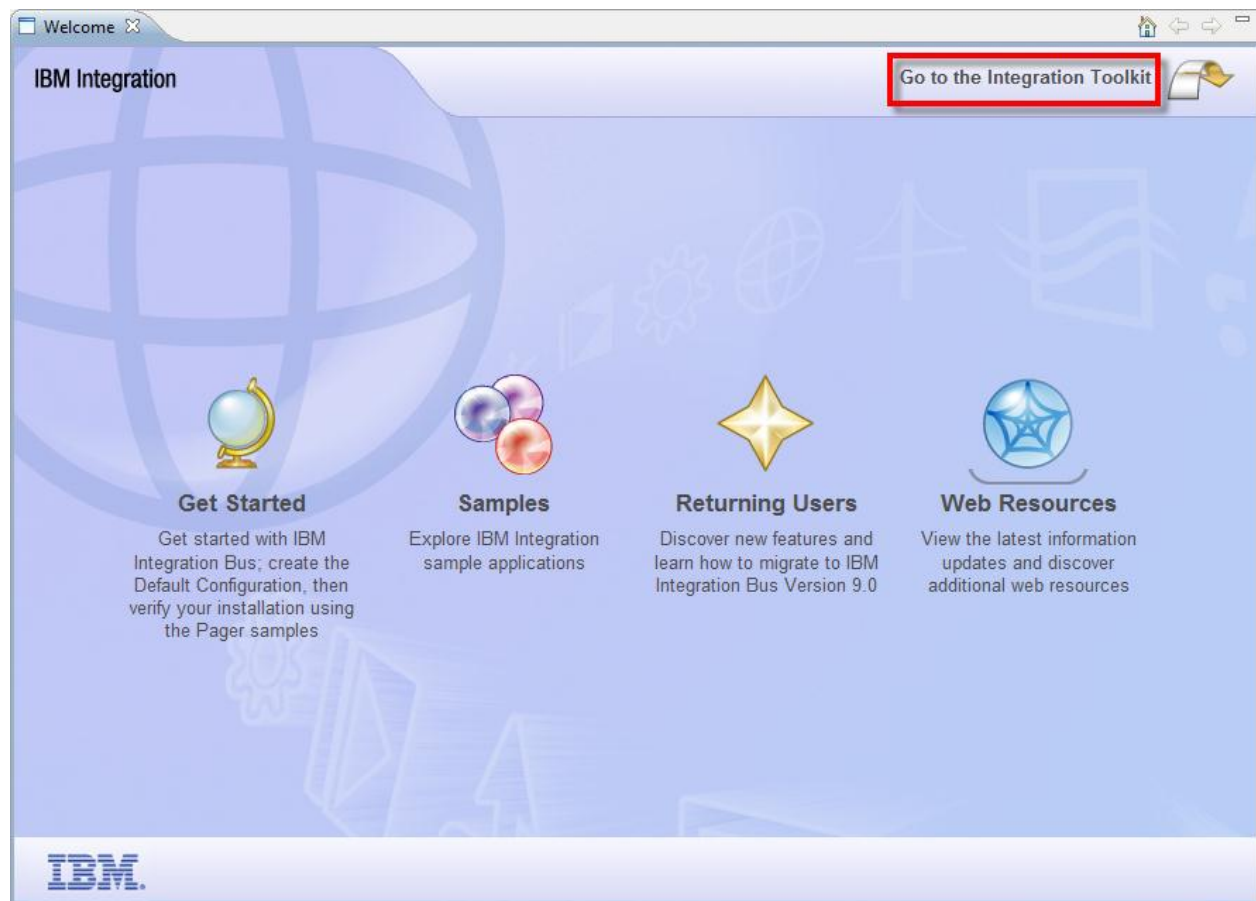


- __2. Select the **C:\student\AppsLibs\workspace** workspace.

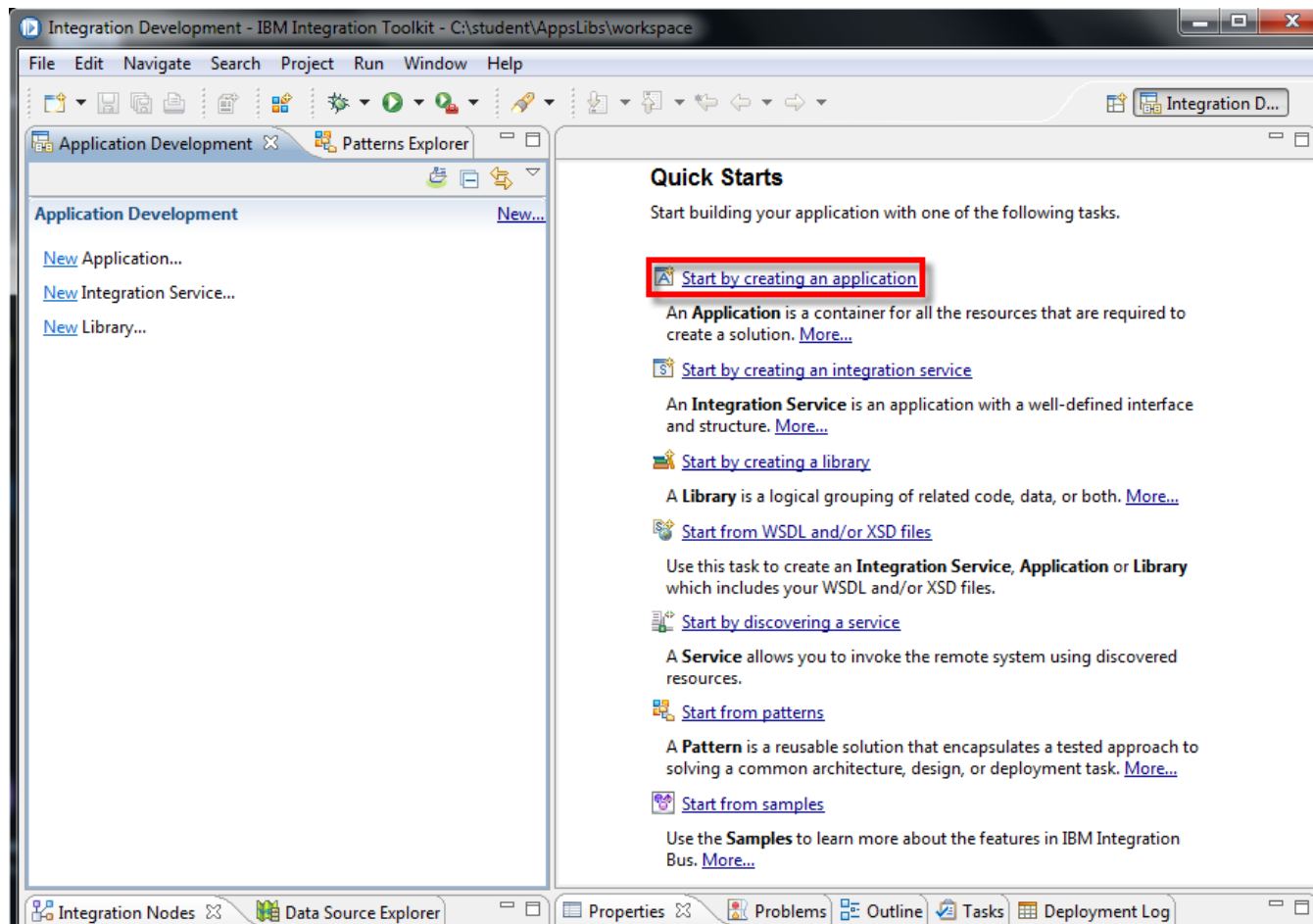


- __3. Press the **OK** button to continue.

- ___4. Click on **Go to the Integration Toolkit**.



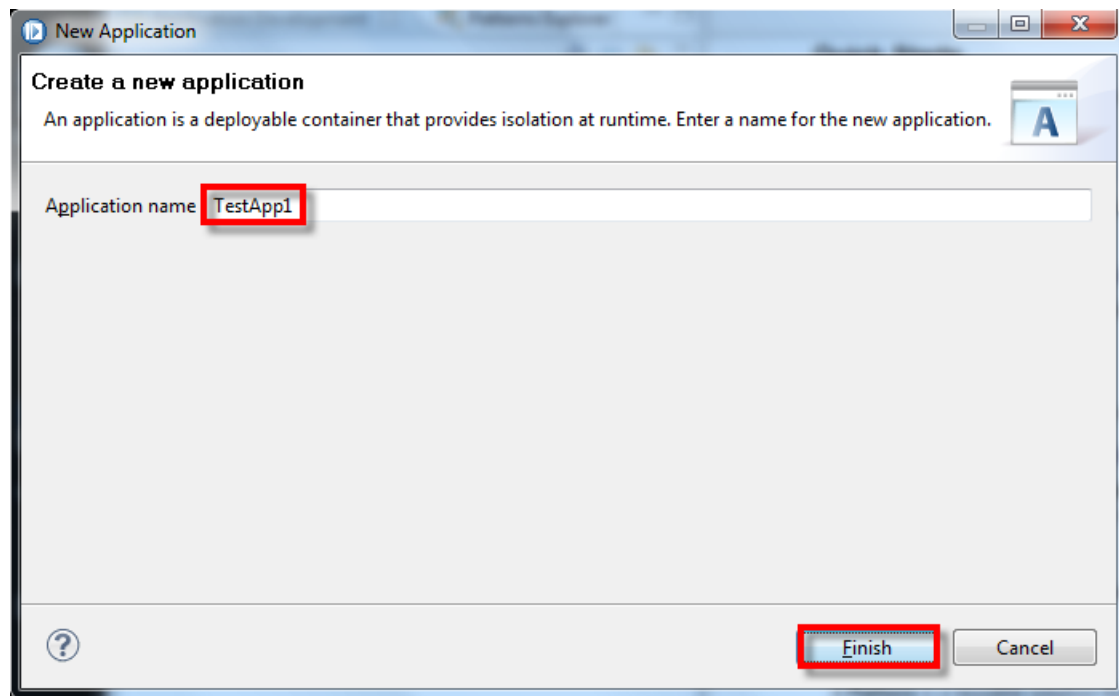
After closing the Welcome Page in the Integration Toolkit, the starting page of a new workspace looks like this.



The default starting point shows the “*Application Development*” view. Note that there are no project options shown in the Application Development view. All *Application* or *Library* or *Pattern* artifacts will be presented as items under the Application Development view.

__5. Create a new Application by clicking **Start by creating an application**.

- __6. Name the new application **TestApp1**.



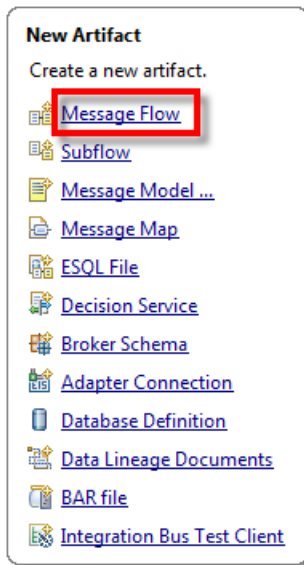
- __7. Press the **Finish** button.

The new application should be visible in the project navigator.

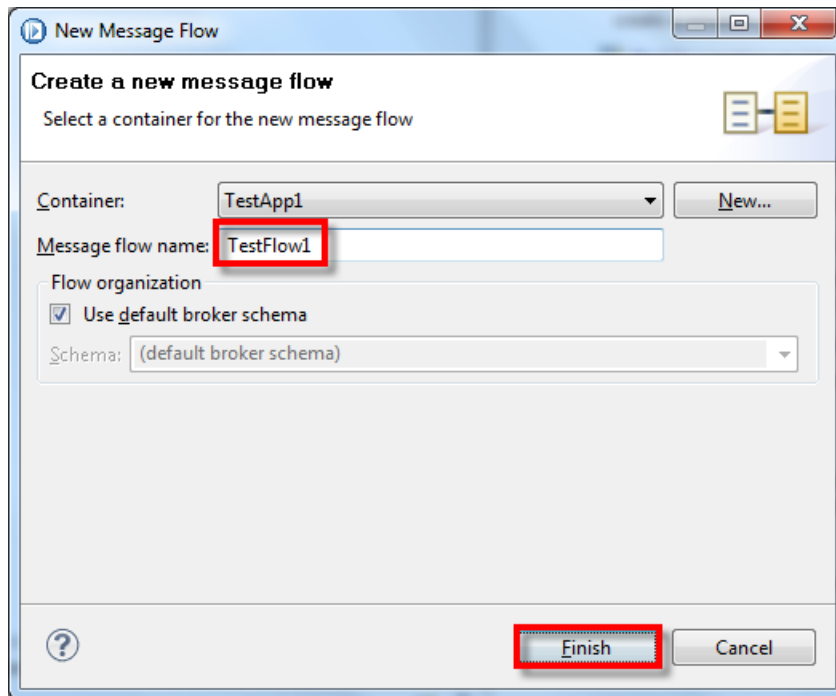


- __8. Click the **New...** hot spot.

__9. Click **Message Flow**.

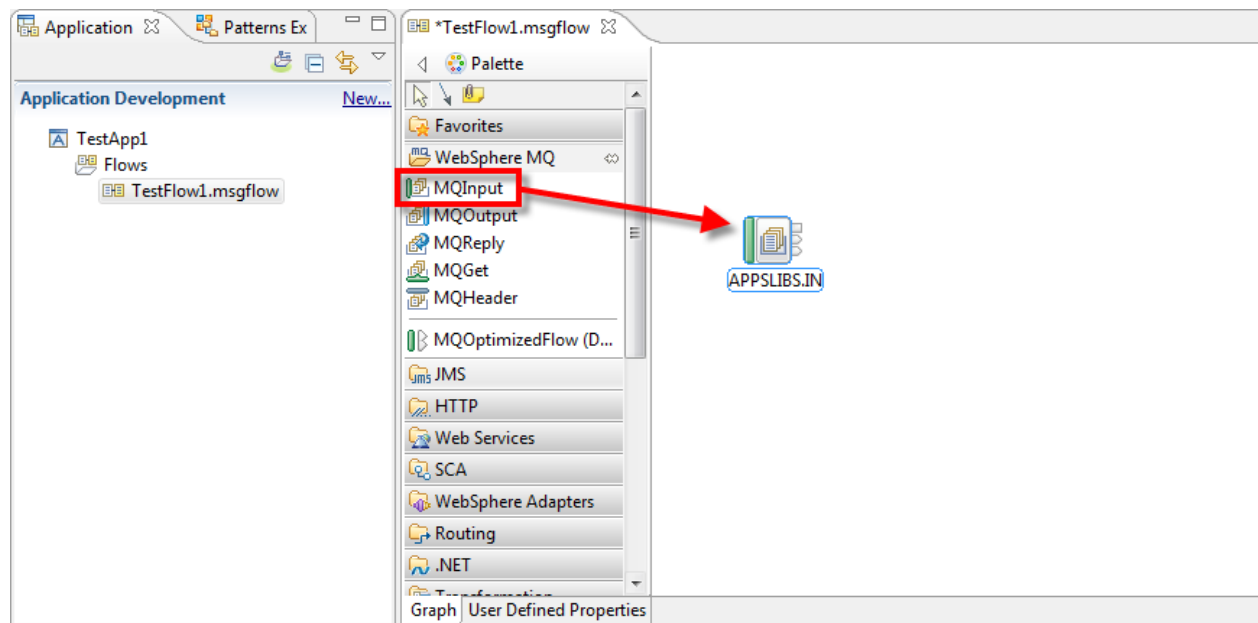


__10. Set the **Message flow name** to **TestFlow1**.



__11. Press the **Finish** button.

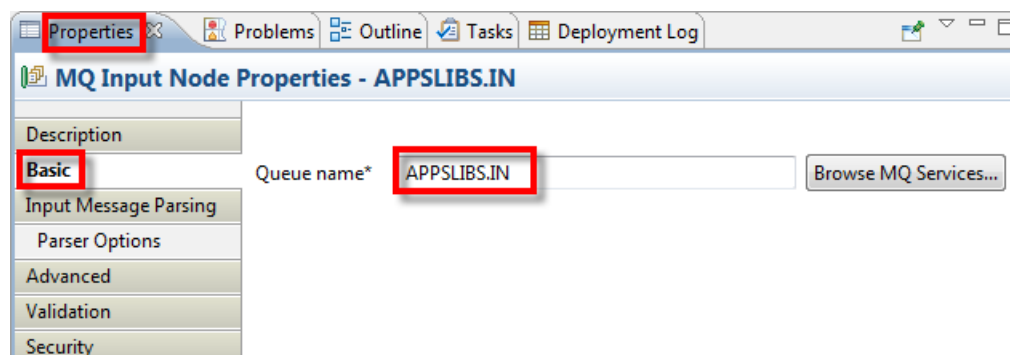
__12. Expand the **WebSphere MQ** folder.



__13. Drag an **MQInput** node to the message flow editor canvas.

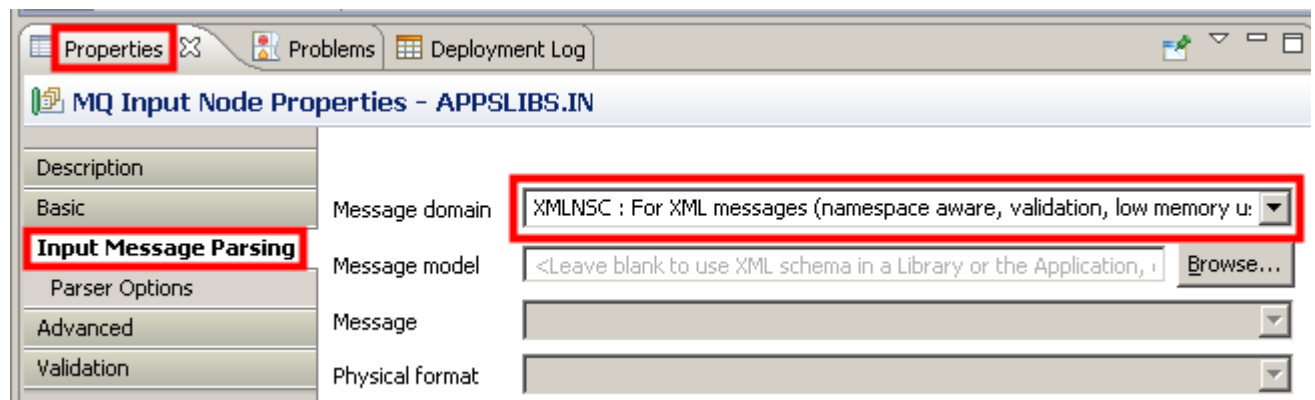
__14. Change the name to **APPSLIBS.IN**.

__15. In the **Properties** pane select the **Basic** tab.



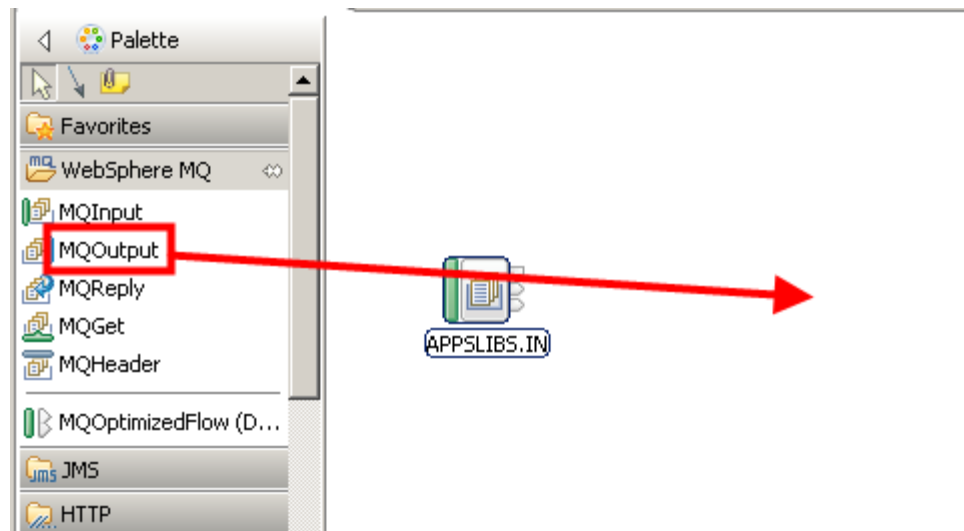
__16. Set the **Queue name** to **APPSLIBS.IN**. Remember that Queue names are case sensitive. All queue names in the labs are upper case.

__17. Select the **Input Message Parsing** tab.



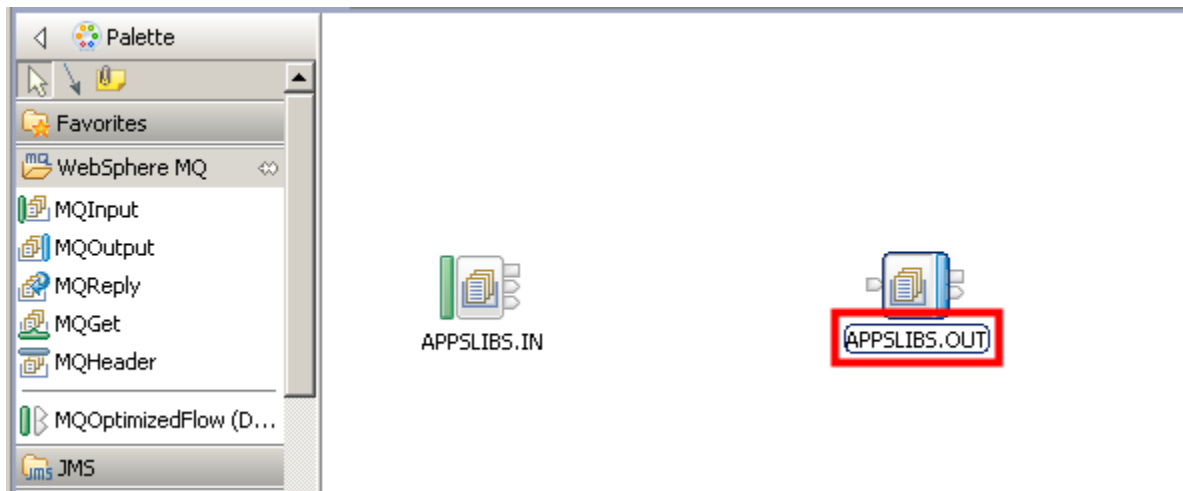
__18. Use the drop down menu to select **XMLNSC** as the **Message domain**.

__19. Select an **MQOutput** node.

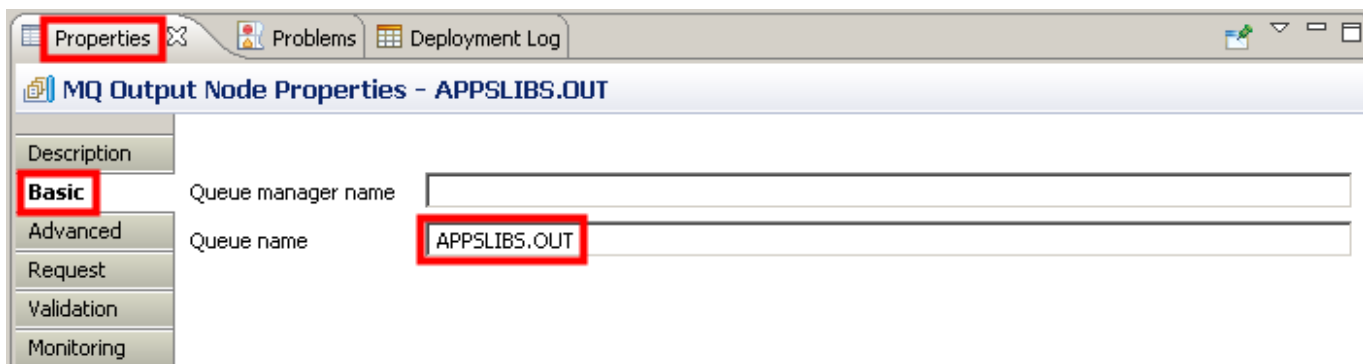


__20. Drag it to the canvas to the right of the **APPSLIBS.IN** node.

__21. Change the name of the node to **APPSLIBS.OUT**.

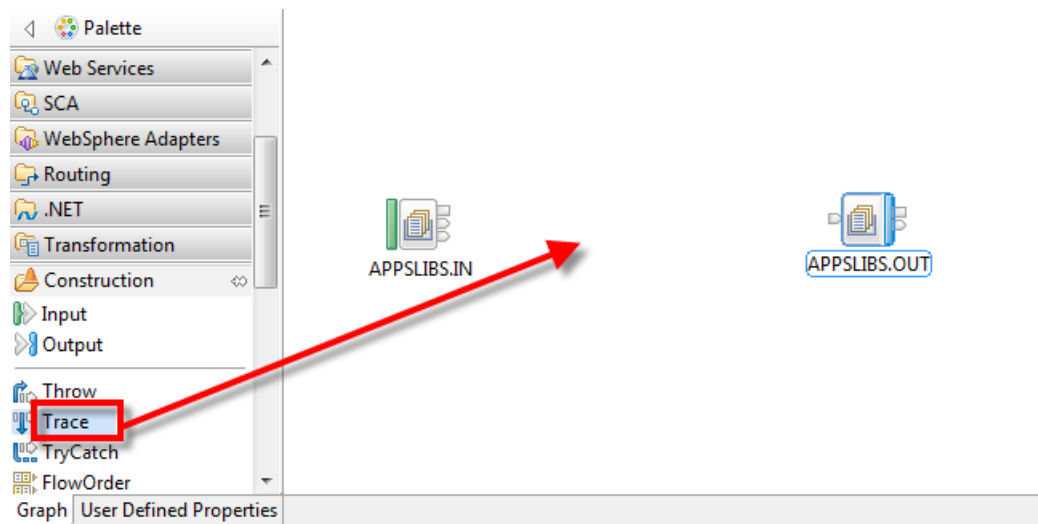


__22. In the **Properties** pane select the **Basic** tab.



__23. Set the **Queue name** to **APPSLIBS.OUT**.

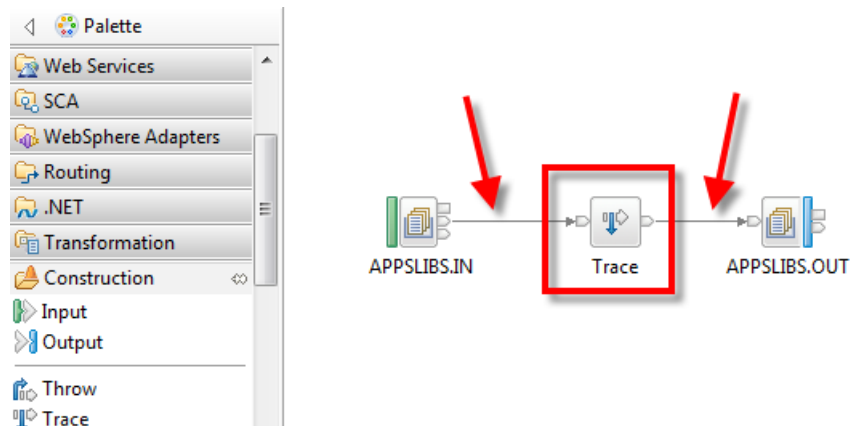
__24. Expand the **Construction** drawer.



__25. Select a **Trace** node.

__26. Place it between the other two nodes.

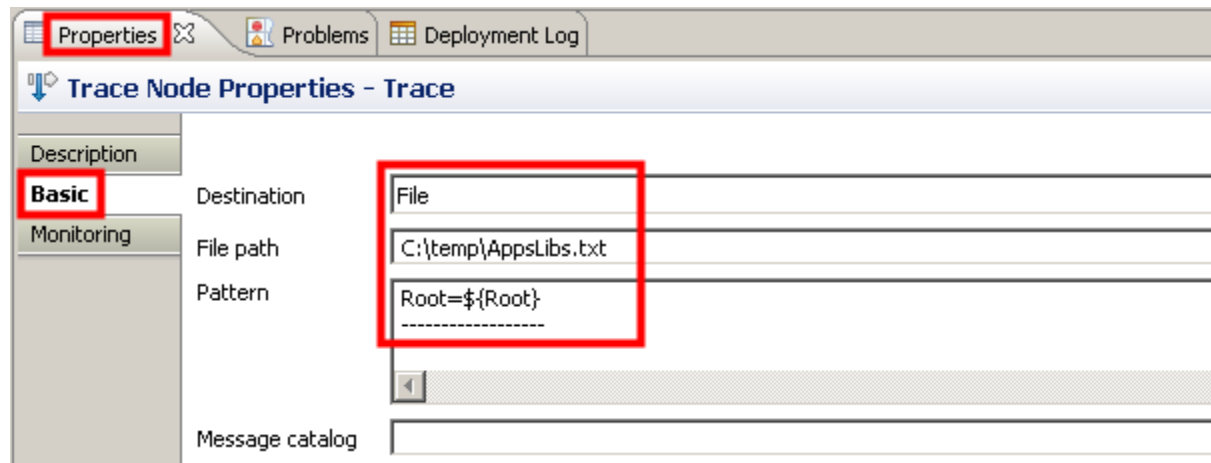
__27. Wire the **Out** terminal of the **APPSLIBS.IN** node to the **In** terminal of the **Trace** node.



__28. Wire the **Out** terminal of the **Trace** node to the **In** terminal of the **APPSLIBS.OUT** node.

__29. Select the **Trace** node.

__30. In the **Properties** pane select the **Basic** tab.



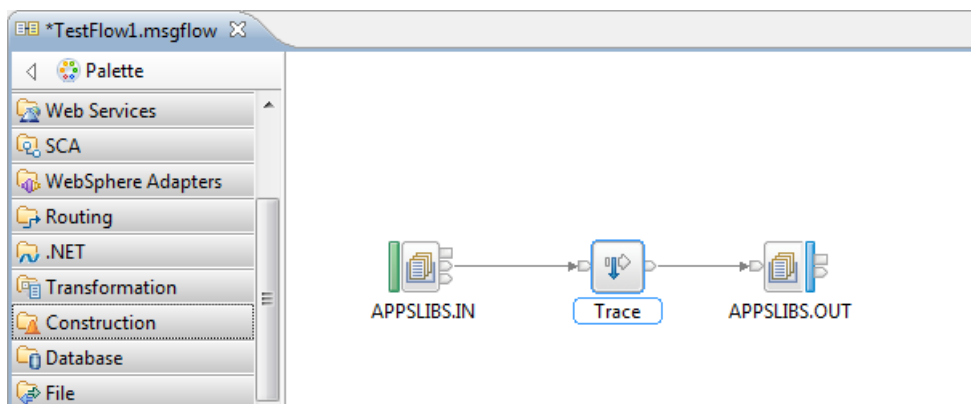
__31. Use the drop down menu to set the **Destination** to **File**.

__32. Set the **File path** to **c:\temp\AppsLibs.txt**.

__33. Set the **Pattern** to the following:

Root=\${Root}

The message flow is now complete.

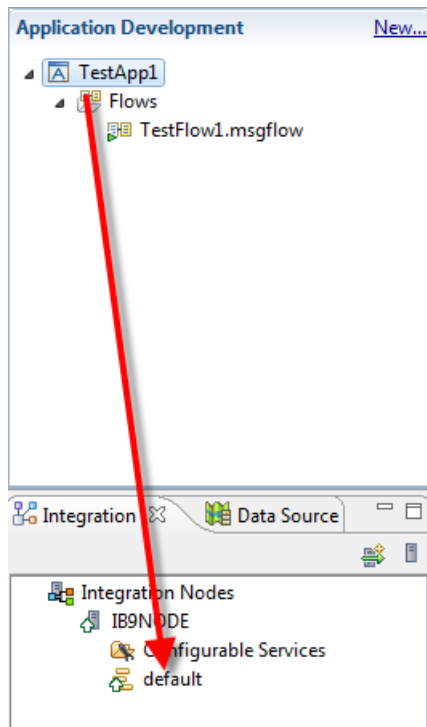


__34. Save the message flow (**Ctrl+S**).



The application can now be deployed using drag and drop.

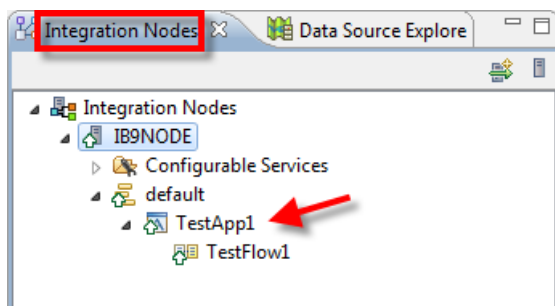
__35. Select the **TestApp1** application.



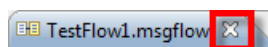
__36. Drag it to the **default** Integration Server.

__37. Drop the **TestApp1** application on the **default** Integration Server.

After the deployment is complete the **TestApp1** application should be visible in the **Integration Nodes** view.



__38. Close the TestApp1 message flow.

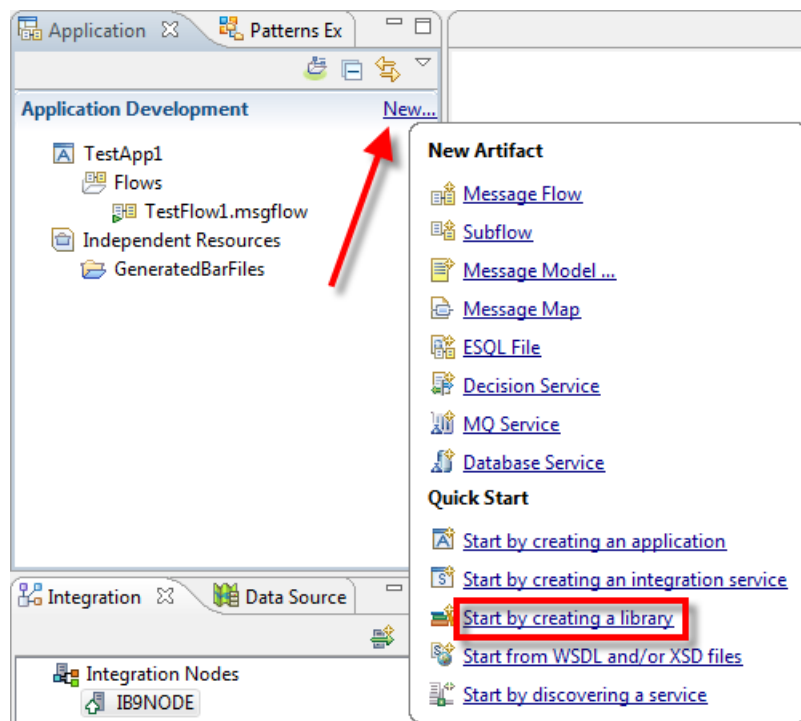


1.3 Creating a library and library references

A library is a container where shared resources can be stored. Message models and common subflows are examples of shared resources that should be stored in libraries. Libraries can be used by more than one application. Applications can contain references to more than one library.

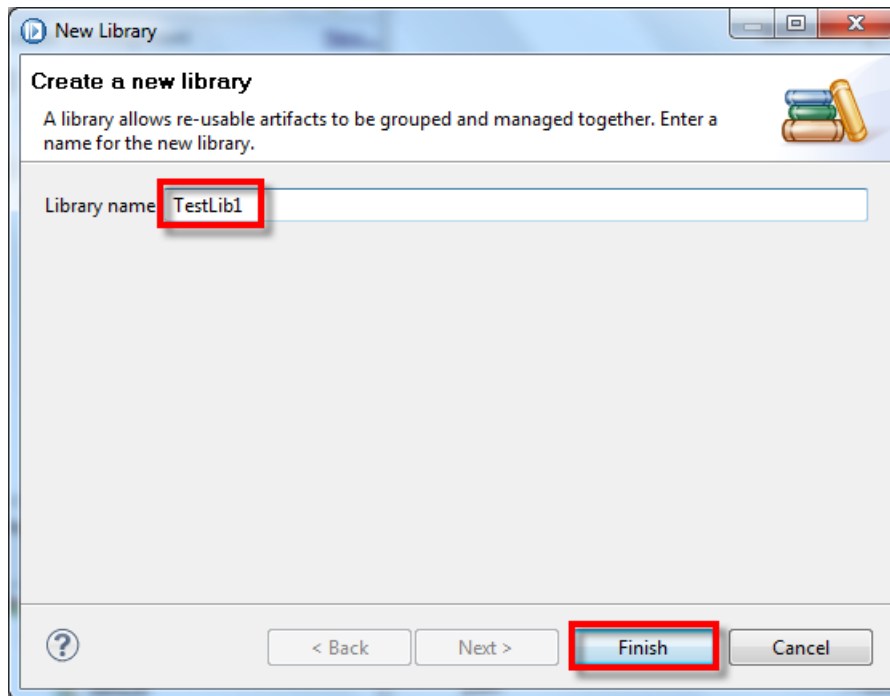
Subflows stored in libraries can be replaced independently of applications. When a shared subflow is replaced it is replaced for all applications that are using the subflow.

- __1. Click the **New...** button.



- __2. Select **Start by creating a library** from the menu.

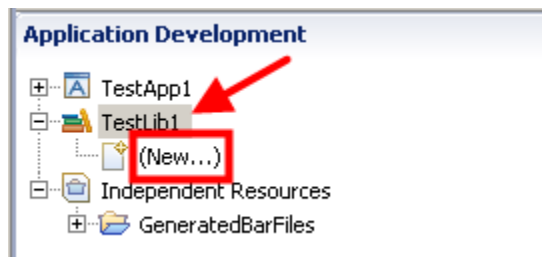
- ___3. Set the **Library name** to **TestLib1**.



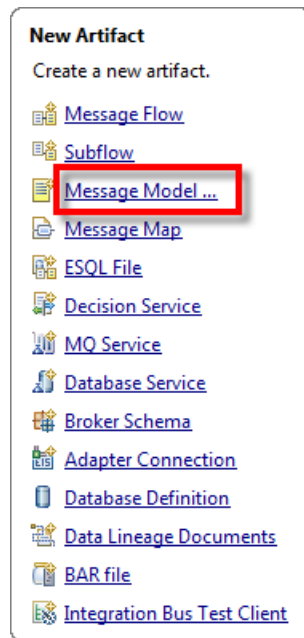
- ___4. Press the **Finish** button to create the library.

The new library should be visible in the project navigator.

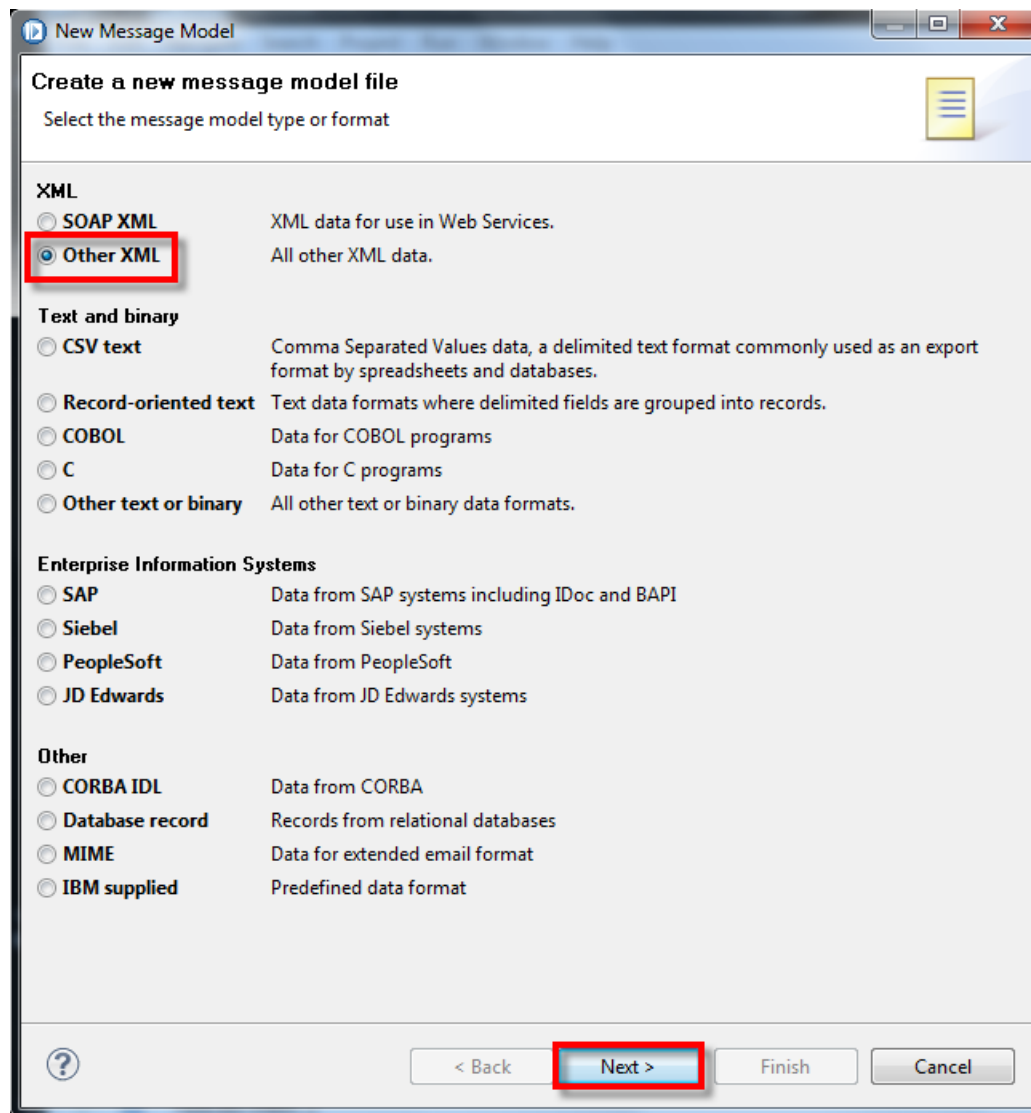
- ___5. Select the **New...** option under the **TestLib1** library.



___6. Select **Message Model** from the menu.

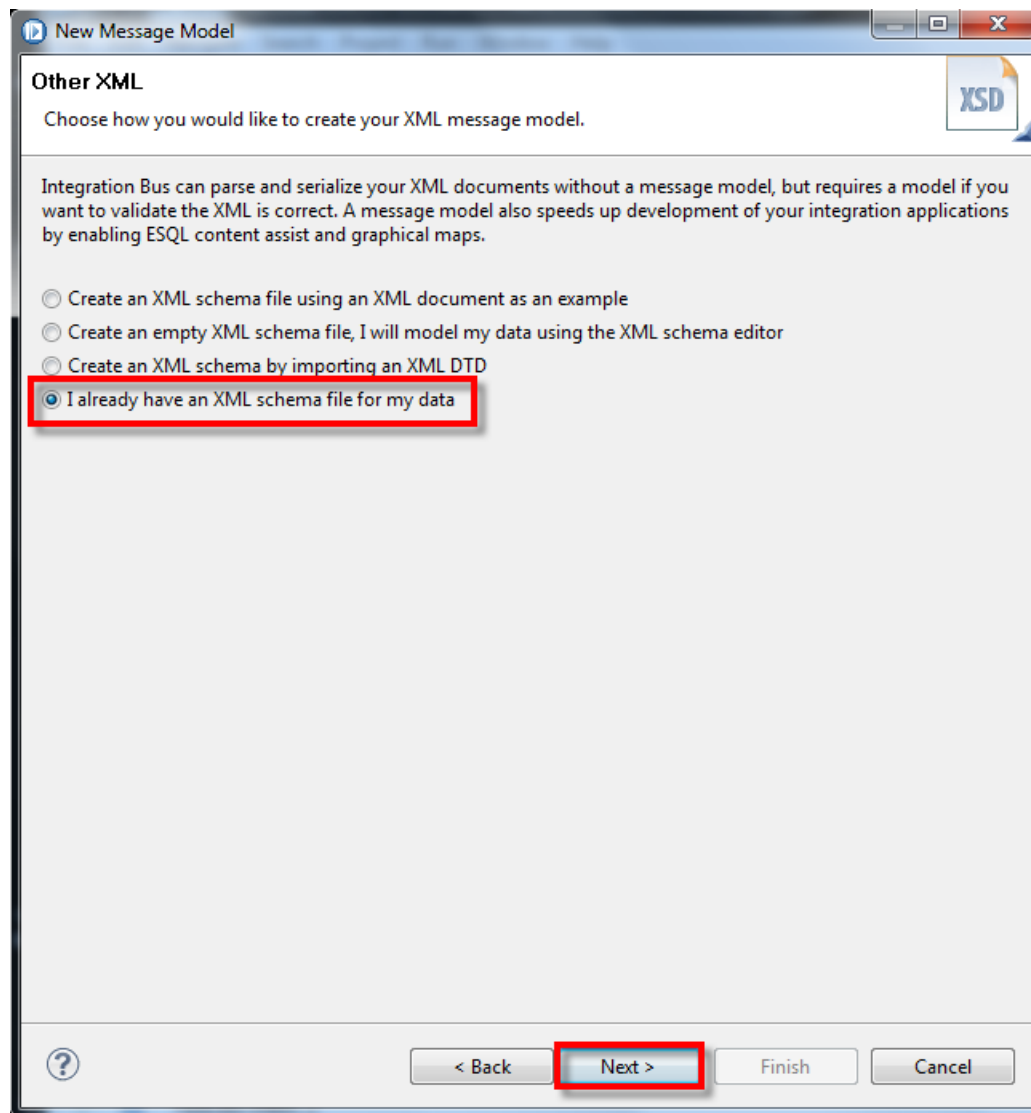


__7. Select the **Other XML** radio button.



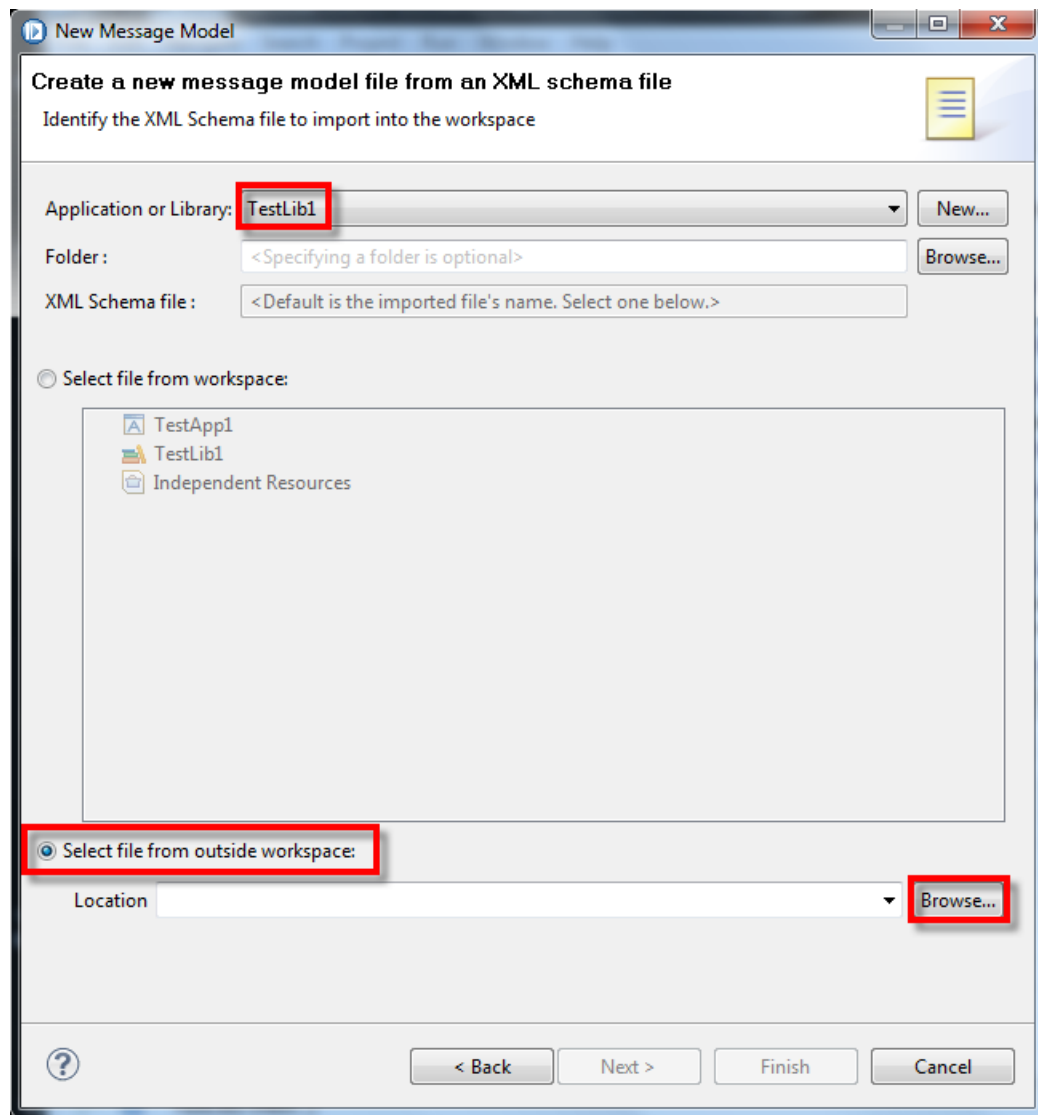
__8. Press the **Next** button to continue.

- __9. Select the **I already have an XML schema file for my data** radio button.



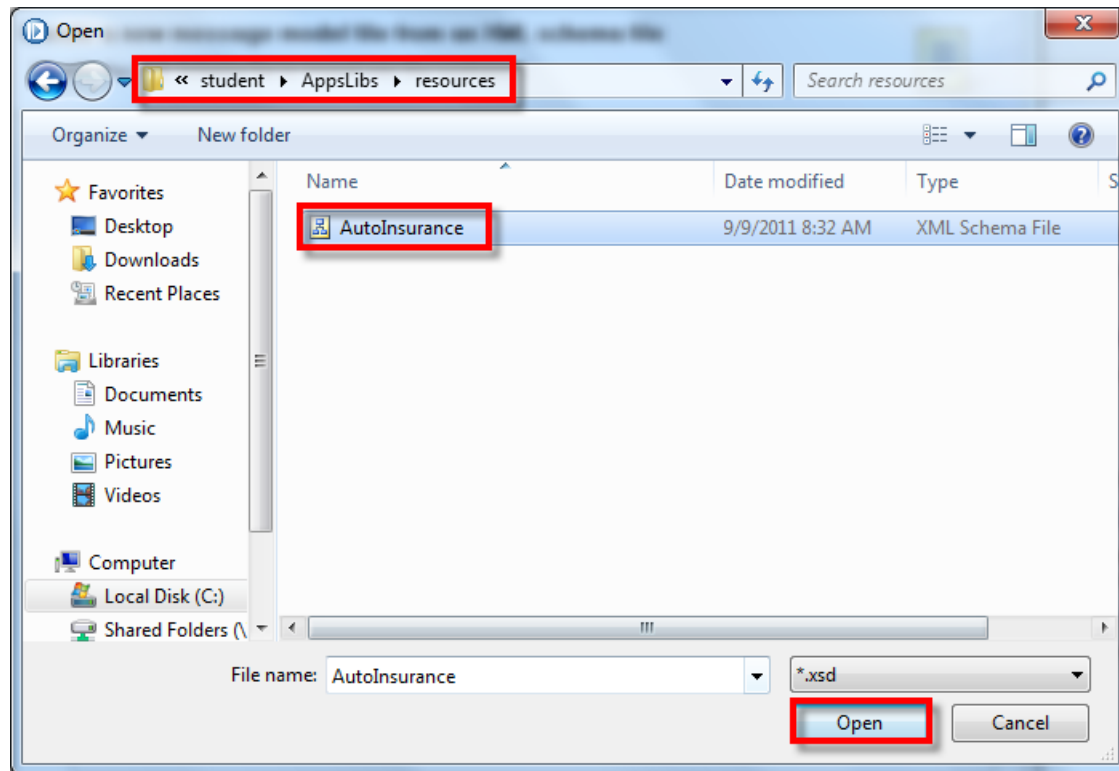
- __10. Press the **Next** button to continue.

- __11. Make sure that the **Application or Library** is set to **TestLib1**.



- __12. Select the **Select file from outside workspace** radio button.
- __13. Press the **Browse...** button.

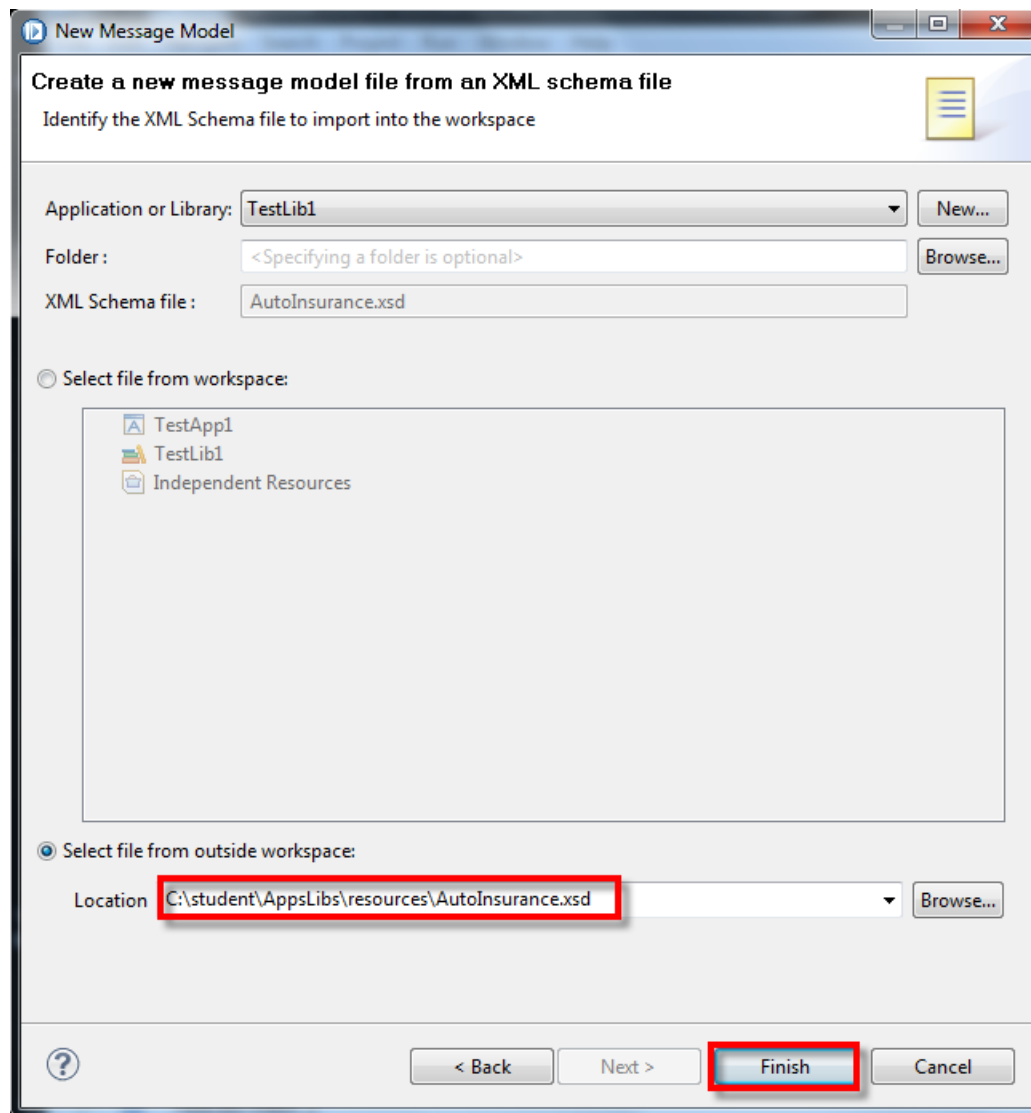
__14. Navigate to the **C:\student\AppsLibs\resources** directory.



__15. Select the **AutoInsurance.xsd** schema file.

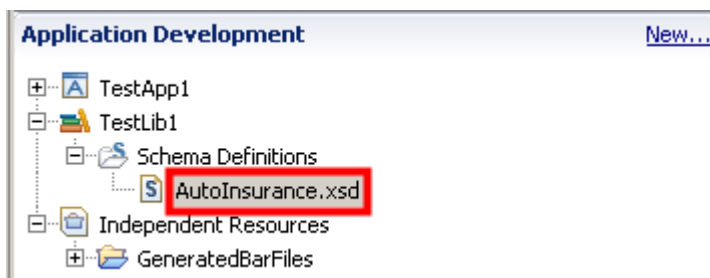
__16. Press the **Open** button.

__17. Made sure the **Location** is filled in.

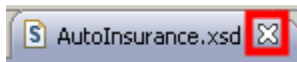


__18. Press the **Finish** button to import the schema file into the library.

The new message model should be visible in the library.

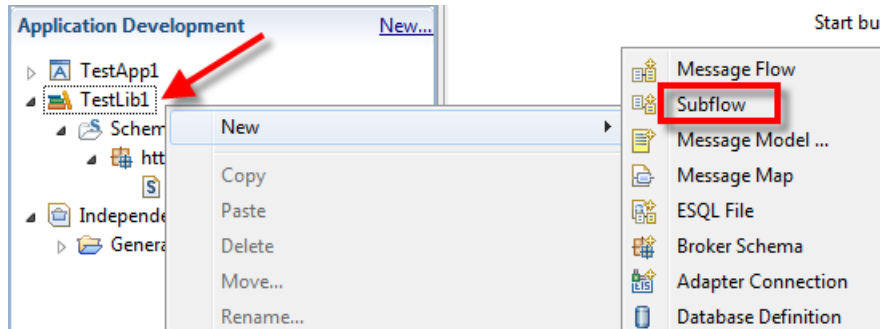


__19. The Schema editor will open. Close it.



A new sub flow will now be added to the library.

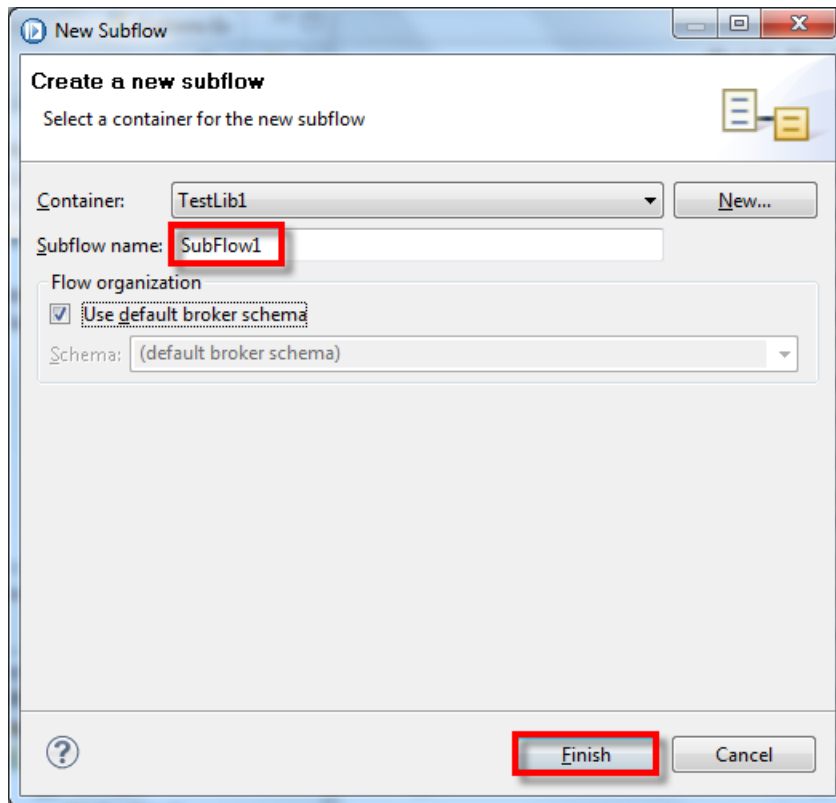
__20. Select the **TestLib1** library.



__21. Press the right mouse button.

__22. Select **New→Subflow** from the menu.

__23. Make sure the **Container** is set to **TestLib1**.

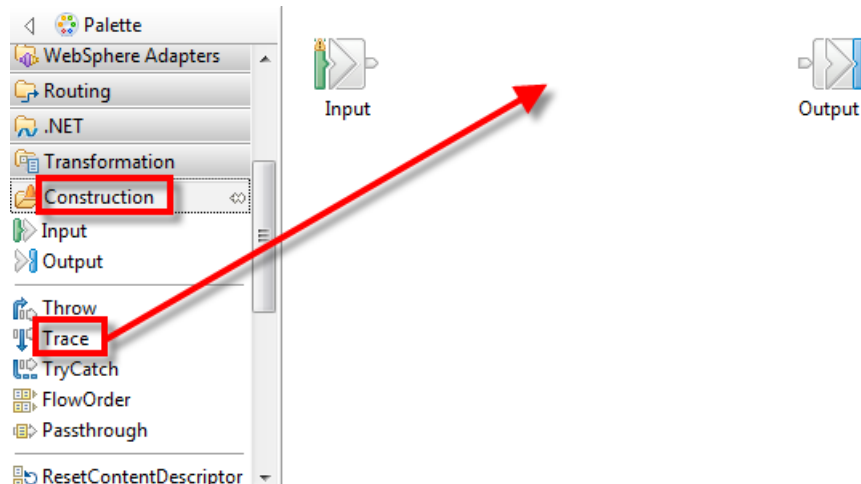


__24. Enter **SubFlow1** as the **Subflow name**.

__25. Press the **Finish** button to create the new subflow.

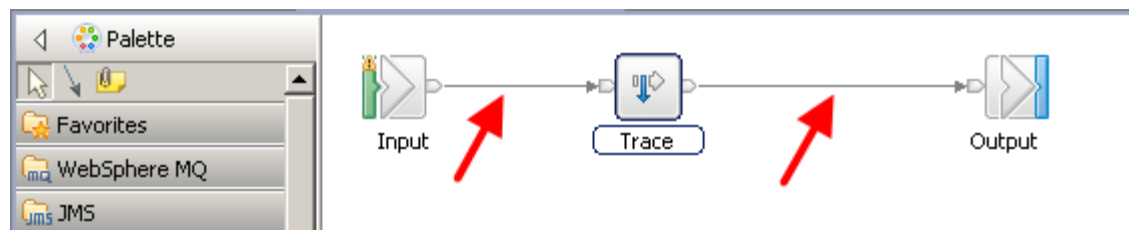
The message flow editor will open automatically. The wizard will automatically populate the new subflow with an input and an output node.

__26. Expand the **Construction** folder.



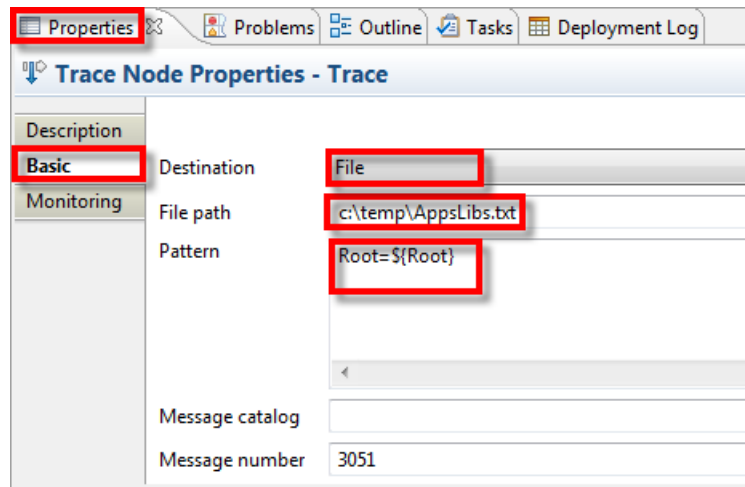
__27. Select a **Trace** node and drag it to a location between the **Input** and **Output** nodes.

__28. Wire the **Out** terminal of the **Input** node to the **In** terminal of the **Trace** node.



__29. Wire the **Out** terminal of the **Trace** node to the **In** terminal of the **Output** node.


__30. In the **Properties** pane select the **Basic** tab.



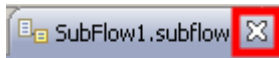
__31. Select a **Destination** of **File** using the drop down list.

__32. Set the **File path** to **c:\temp\AppsLibs.txt**.

__33. Enter **Root=\${Root}** as the **Pattern**.

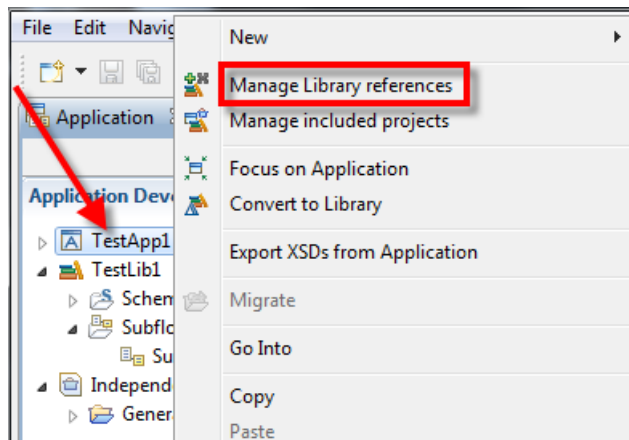
__34. Save the message flow (**Ctrl+S**). 

__35. Close the message flow editor.



A library reference will now be added to the **TestApp1** application.

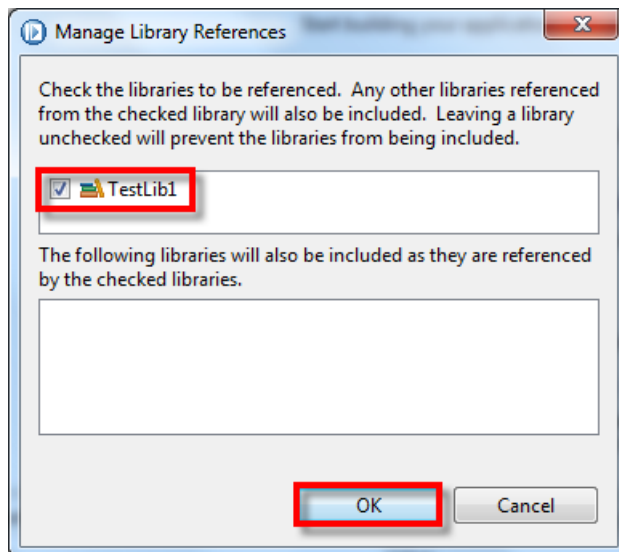
__36. Select the **TestApp1** application.



__37. Press the right mouse button.

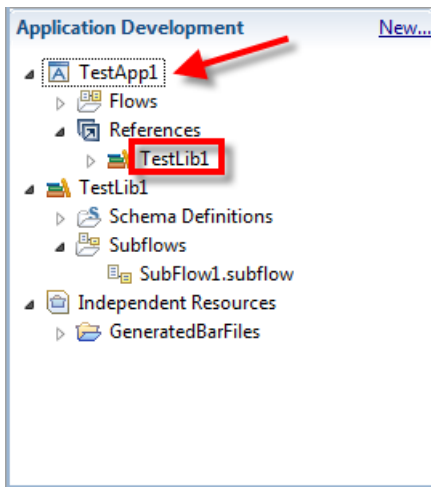
__38. Select **Manage Library references** from the menu.

__39. Select the **TestLib1** check box.

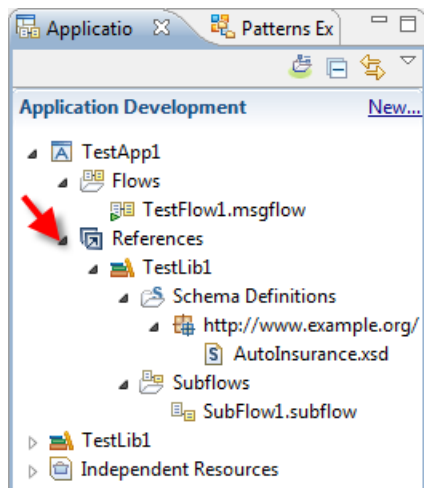


- __40. Press the **OK** button to create the library reference.

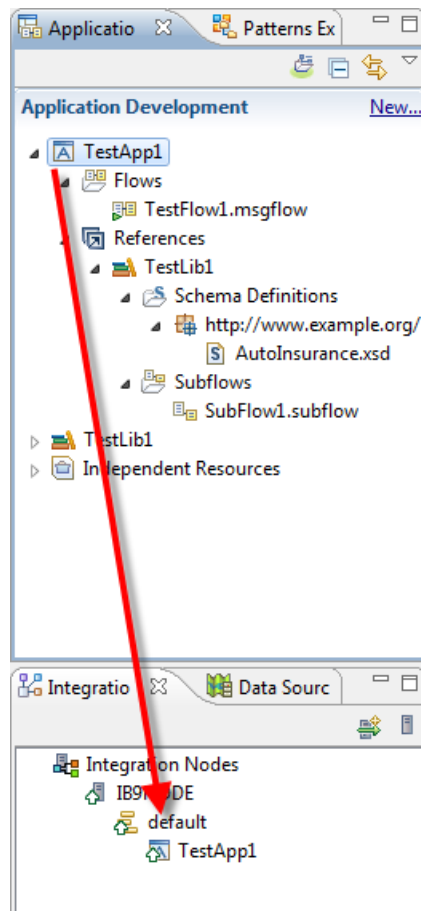
The library reference should now be visible in the **TestApp1** application.



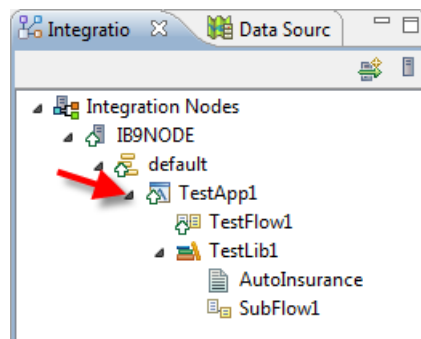
- __41. Click on the twistie to open up and see the references from **TestLib1**.



- __42. Now let's redeploy the TestApp1 application to the runtime. Select the **TestApp1** application, and then drag and drop it to the **default** Integration Server.



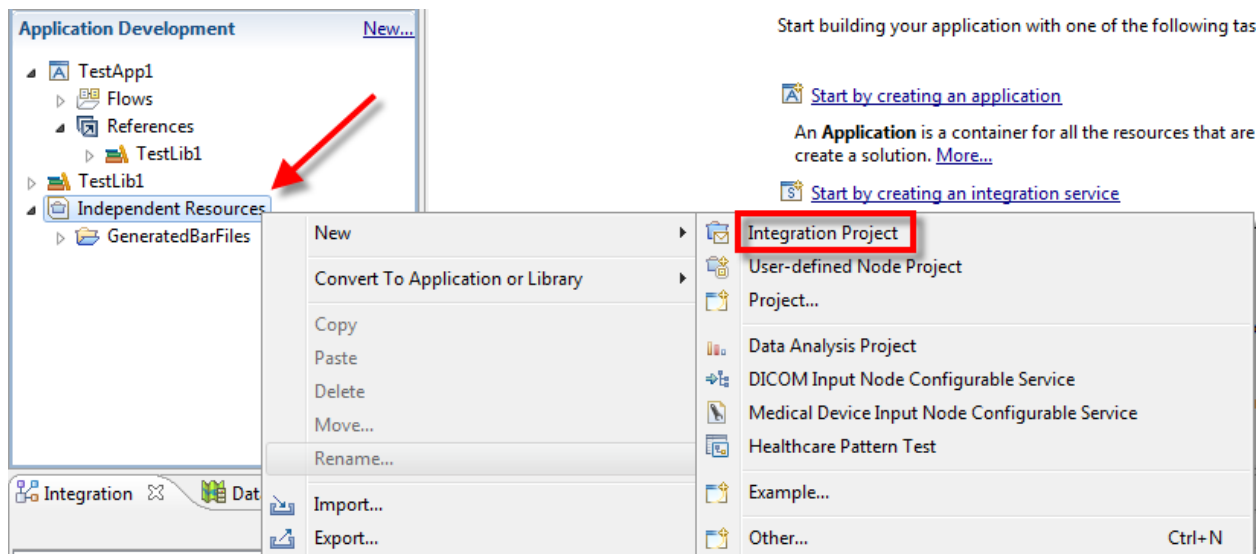
- __43. Once it is deployed, open the twistie on the **TestApp1** application under the **default** Integration Server to see how the TestLib1 library also gets deployed.



1.4 The independent resources view

Not all Integration Bus projects need be part of an application and library. Integration Bus projects that are not part of an application or library are displayed as independent resources.

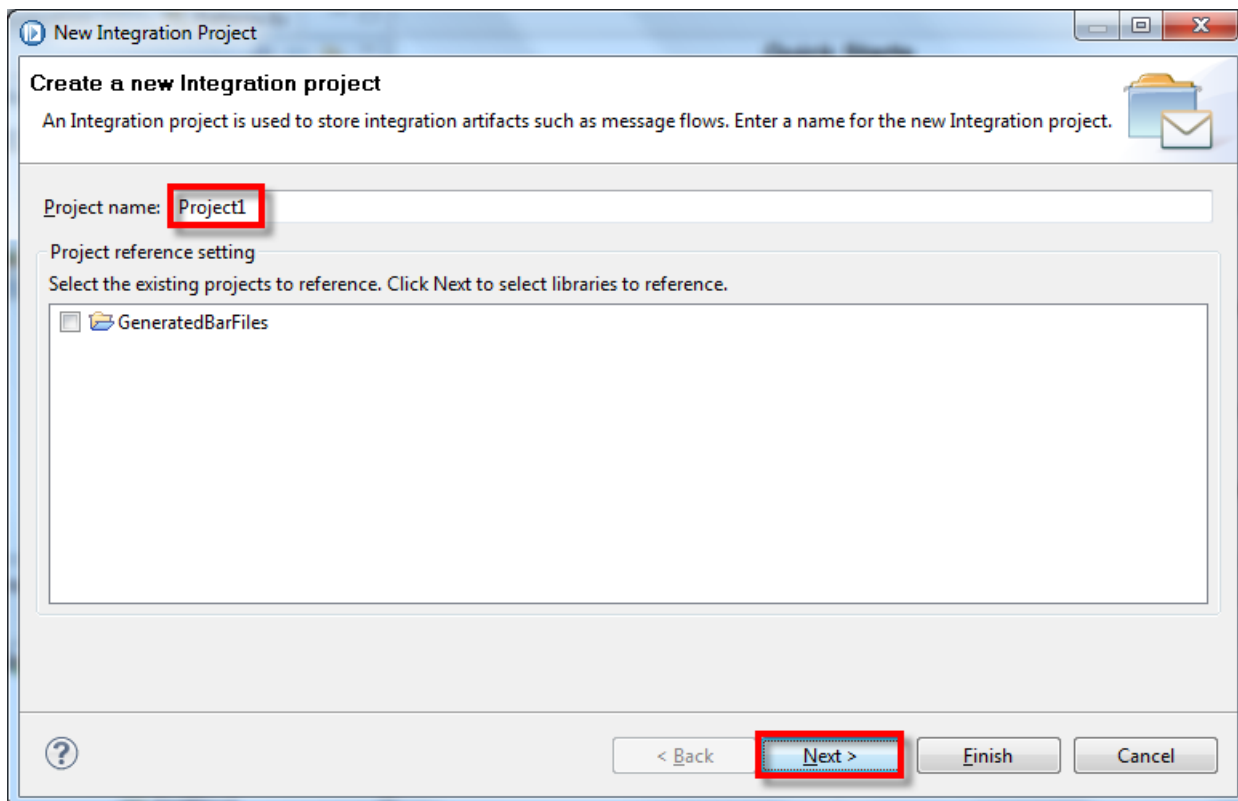
- ___1. Select the **Independent Resources** folder.



- ___2. Press the right mouse button.

__3. Select **New→Integration Project**.

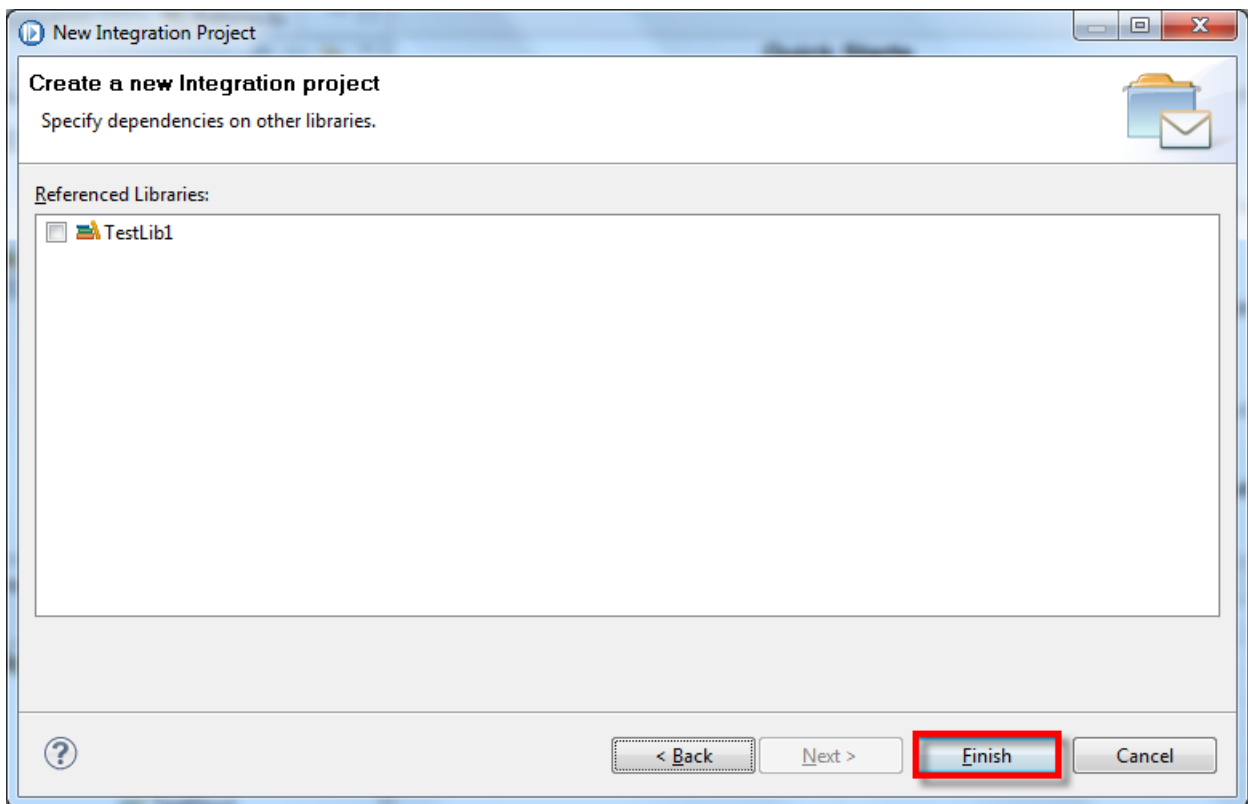
No references to existing independent Integration projects are required.



__4. Enter **Project1** as the **Project name**.

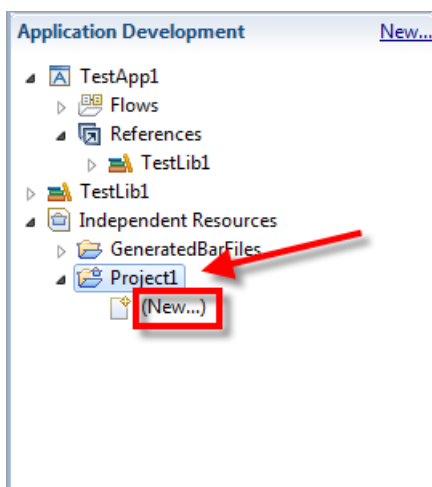
- __5. Press the **Next** button to continue.

No library references are required.



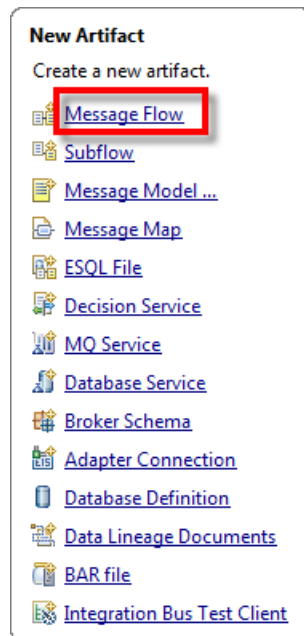
- __6. Press the **Finish** button to create the project.

The new project should be visible in the project navigator.

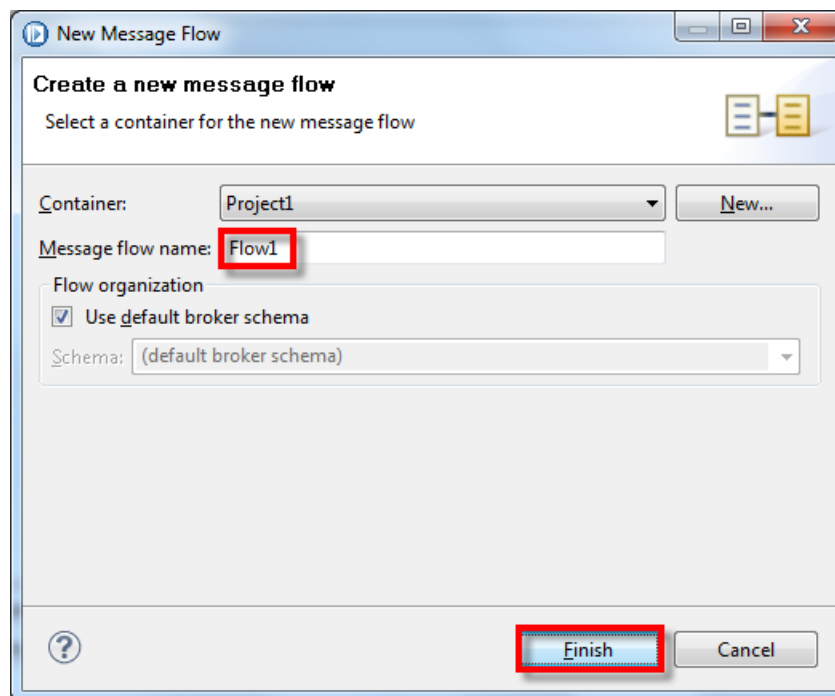


- __7. Click on the **New...** item under **Project1**.

- __8. Select **Message Flow** from the menu.



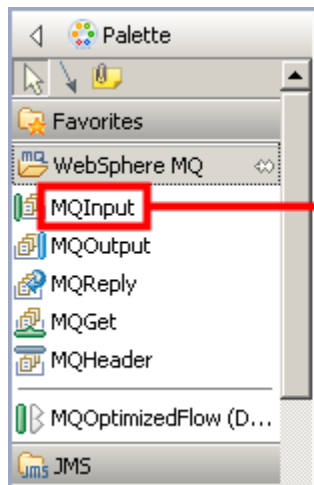
- __9. Enter **Flow1** as the **Message flow name**.



- __10. Press the **Finish** button to create the flow.

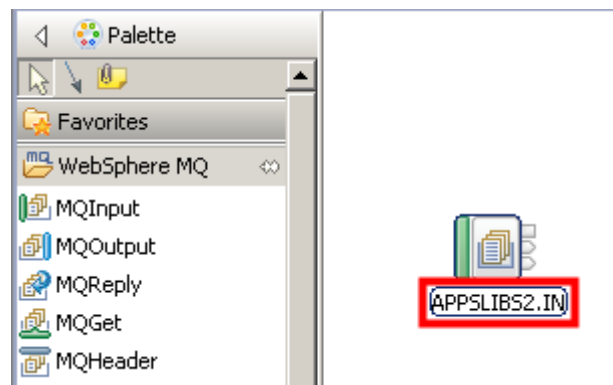
The message flow editor should open. A simple message flow will be constructed.

- __11. Expand the **WebSphere MQ** folder.



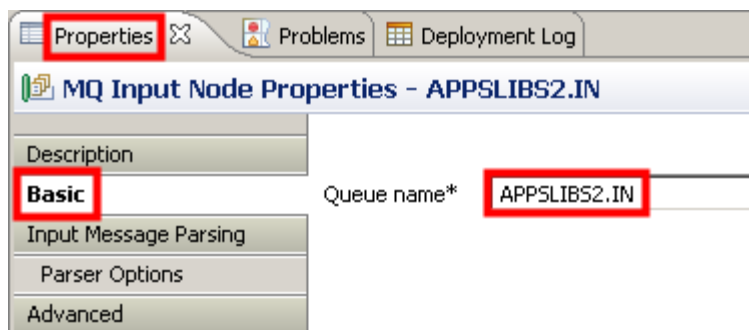
- __12. Select an **MQInput** node from the palette.

- __13. Drag the node onto the canvas.



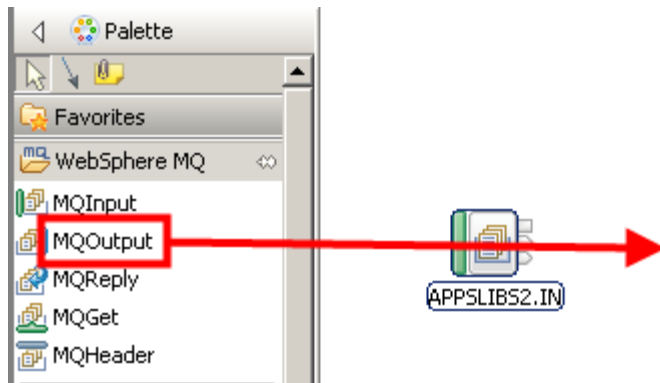
- __14. Change the name of the node to **APPSLIBS2.IN**.

- __15. In the **Properties** pane select the **Basic** tab.

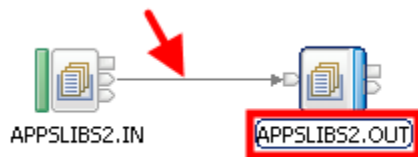


- __16. Set the **Queue name** to **APPSLIBS2.IN**.

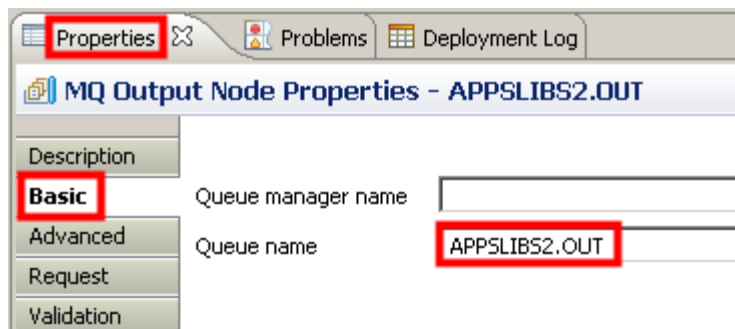
- __17. Select an **MQOutput** node from the palette.




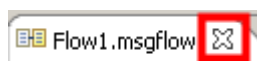
- __18. Drag it to the right of the **APPSLIBS2.IN** node.
- __19. Change the name of the new node to **APPSLIBS2.OUT**.
- __20. Wire the **Out** terminal of the **APPSLIBS2.IN** node to the **In** terminal of the **APPSLIBS2.OUT** node.



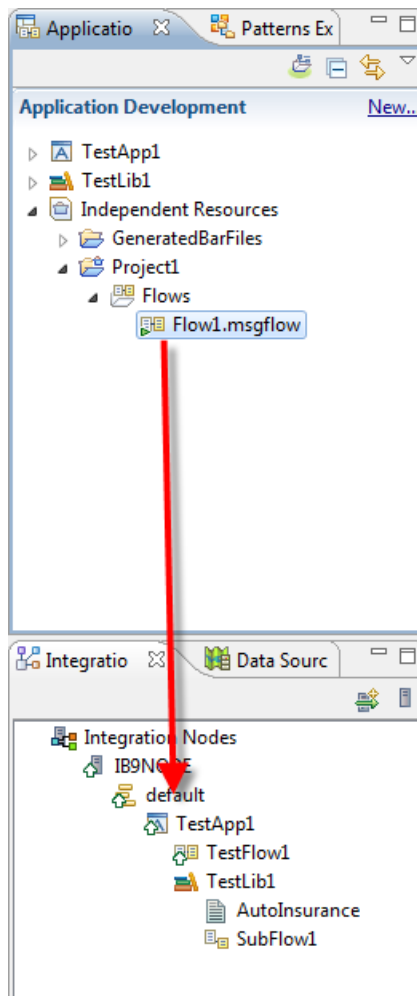
- __21. In the **Properties** pane select the **Basic** tab.



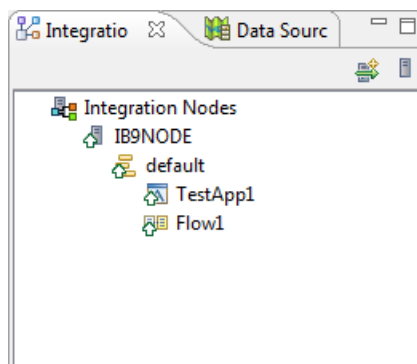
- __22. Enter **APPSLIBS2.OUT** as the **Queue name**.
- __23. Save the message flow (**Ctrl+S**). 
- __24. Close the message flow editor.



- __25. We will now deploy this message flow. Select the Flow1.msgflow and drag and drop it on the default Integration Server.



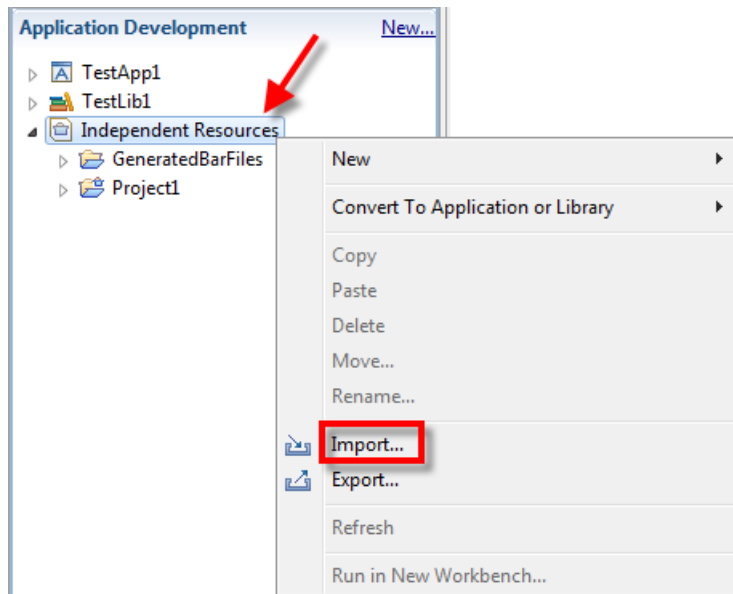
- __26. Notice how it is the **Flow1** message flow alone that is deployed, and that the flow is *not* in an application or library container like TestApp1.



1.5 Working with existing projects

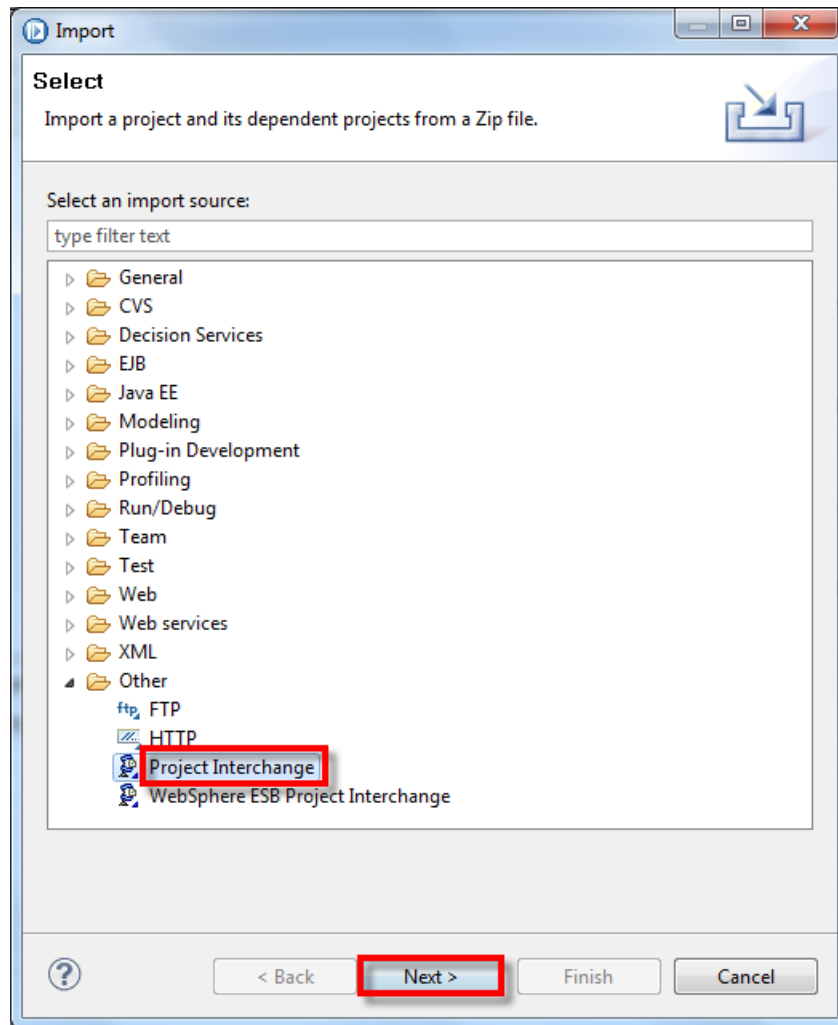
Message flow and message set projects that were created using earlier versions of IBM WebSphere Message Broker can be used in the normal way with the version 9 toolkit. Project interchange files created with earlier versions of the broker toolkit can be imported using the version 9 toolkit. Any such projects will be visible under the independent resources view in the version 9 toolkit.

- __1. In the project navigator pane, select the **Independent Resources** folder.



- __2. Press the right mouse button.
- __3. Select **Import** from the menu.

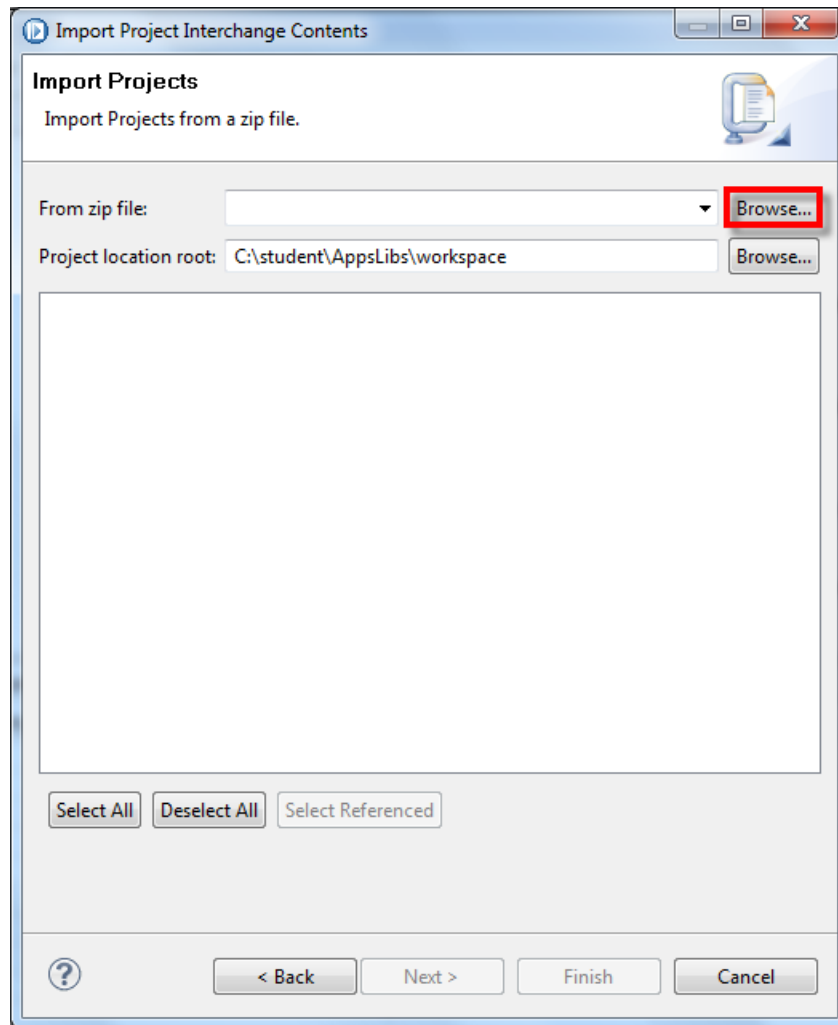
- __4. Expand the **Other** folder.



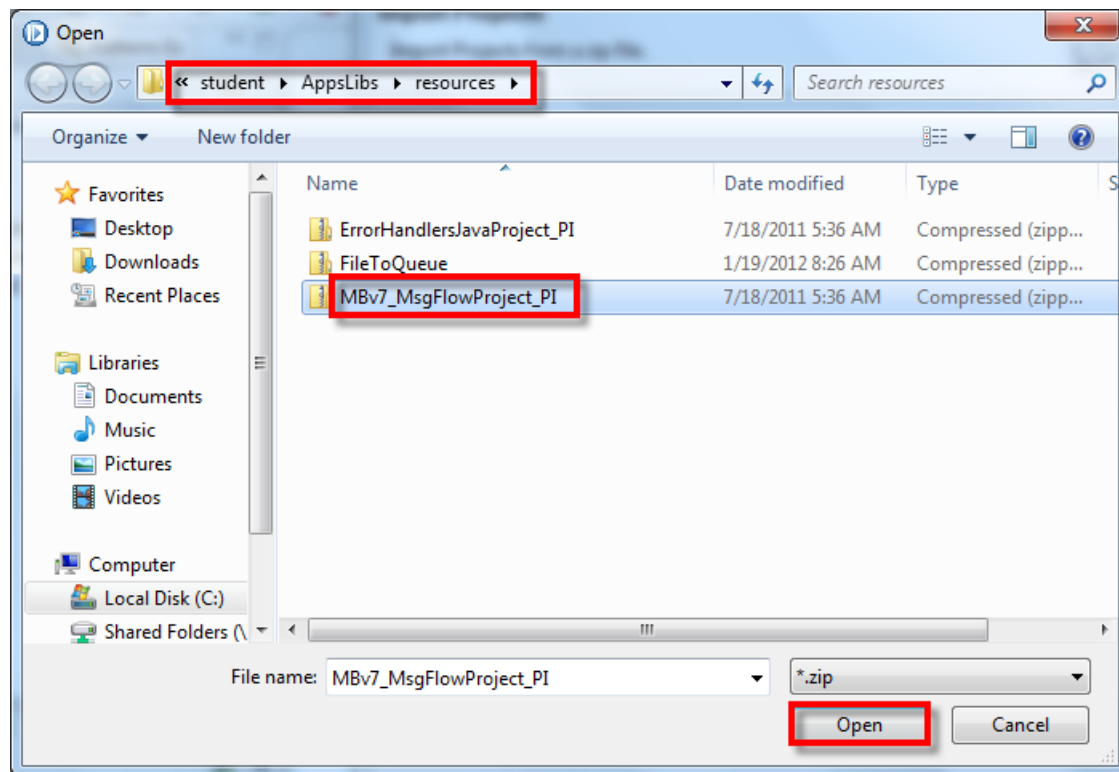
- __5. Select **Project Interchange** as the type of import.

- __6. Press the **Next** button.

- ___7. Press the **Browse** button to locate the zip file.



__8. Navigate to the **C:\student\AppsLibs\resources** directory.

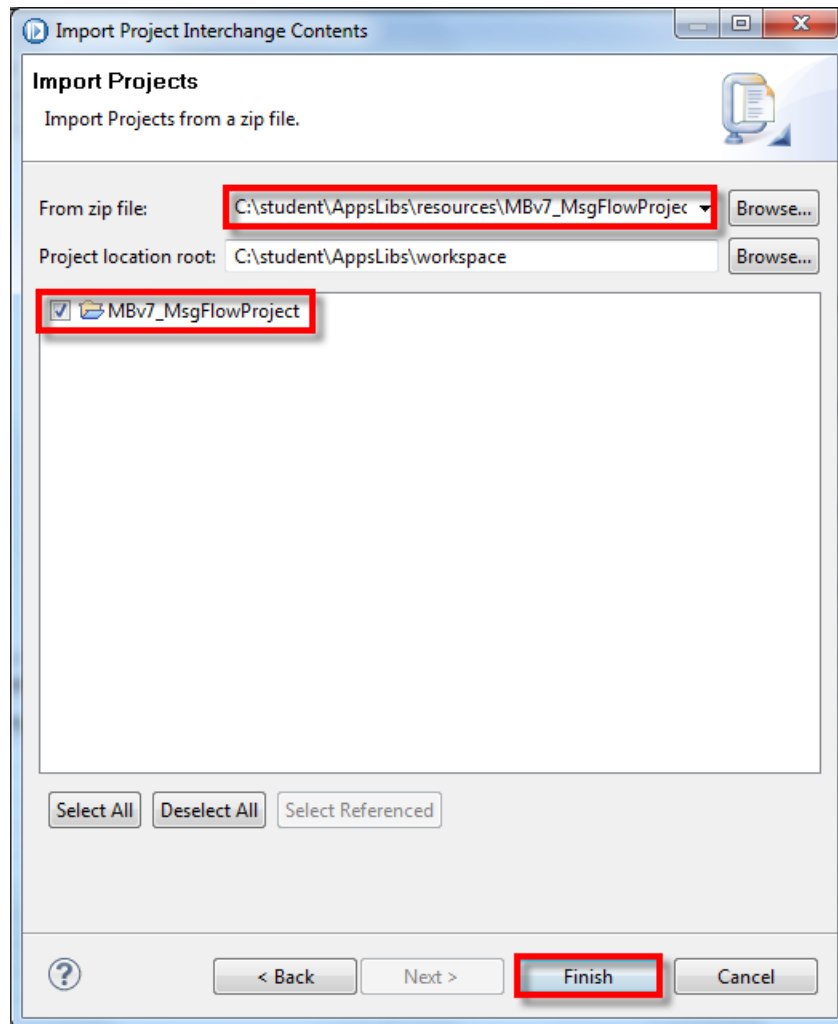


__9. Select the **MBv7_MsgFlowProject_PI.zip** file.

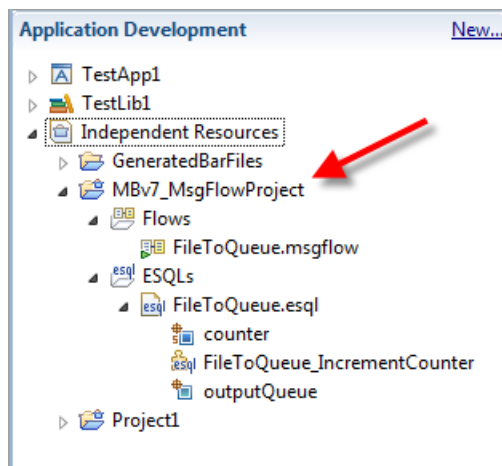
__10. Press the **Open** button.

The zip file should now be visible in the **From zip file** box. The **MBv7_MsgFlowProject** project should be selected.

__11. Press the **Finish** button to start the import.

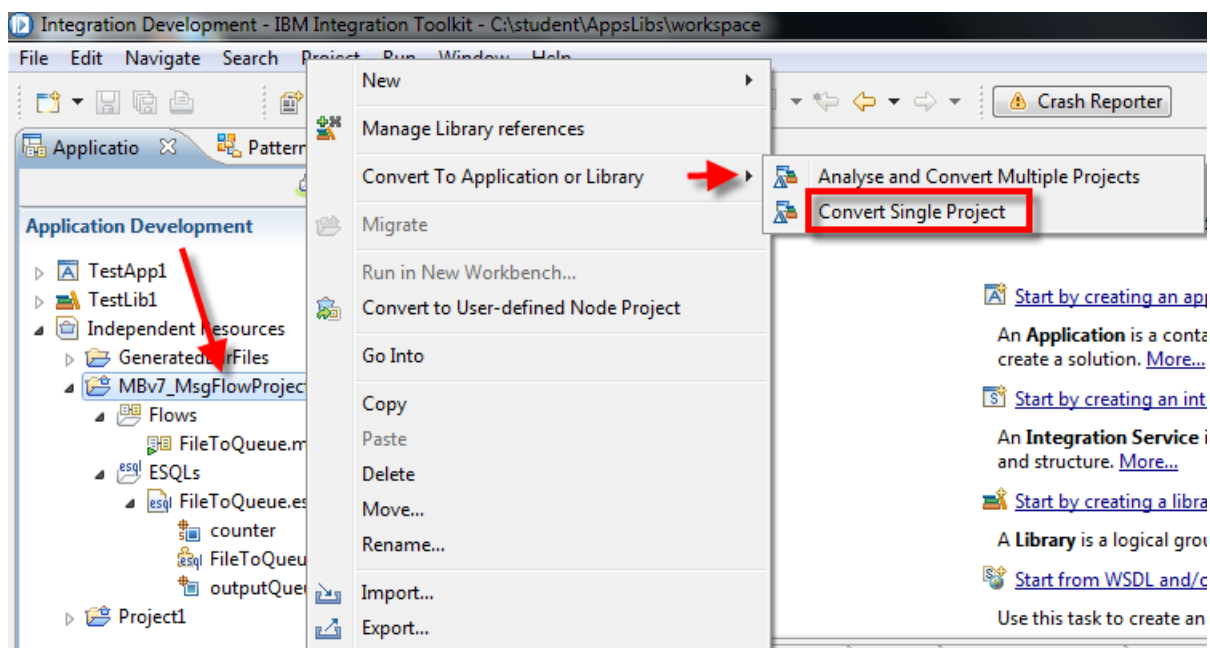


- __12. Fully expand the imported project. It contains a message flow and an ESQL file.



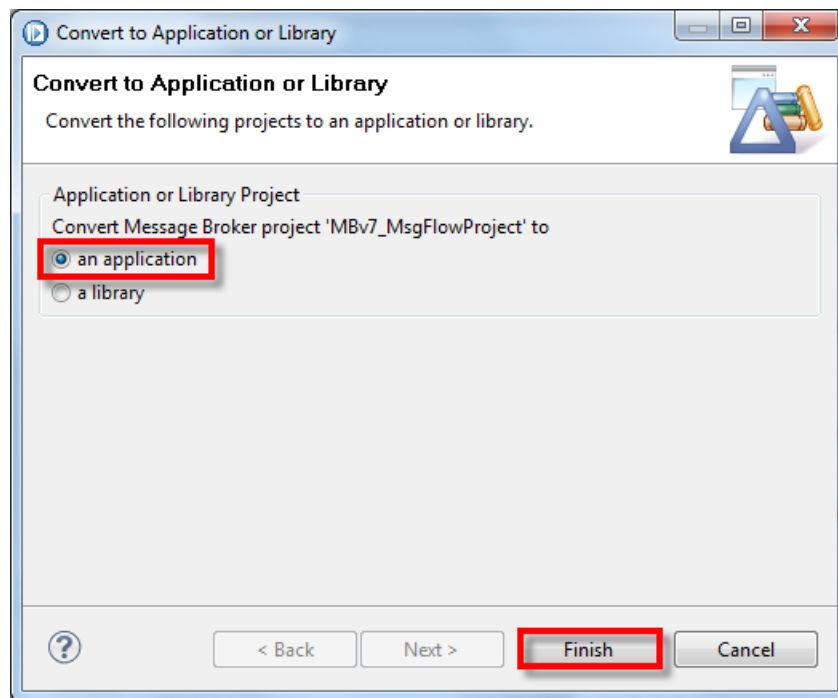
The imported project will now be converted to an application.

- __13. Select the **MBv7_MsgFlowProject** message flow project in the **Independent Resources** view.



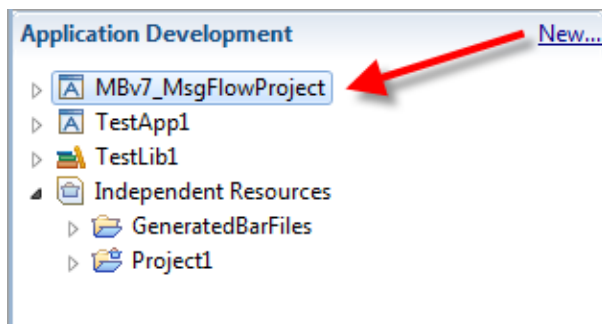
- __14. Press the right mouse button.
- __15. Select **Convert to Application or Library**→**Convert Single Project** from the menu.

__16. Select the **an application** radio button.



__17. Press the **Finish** button to start the conversion.

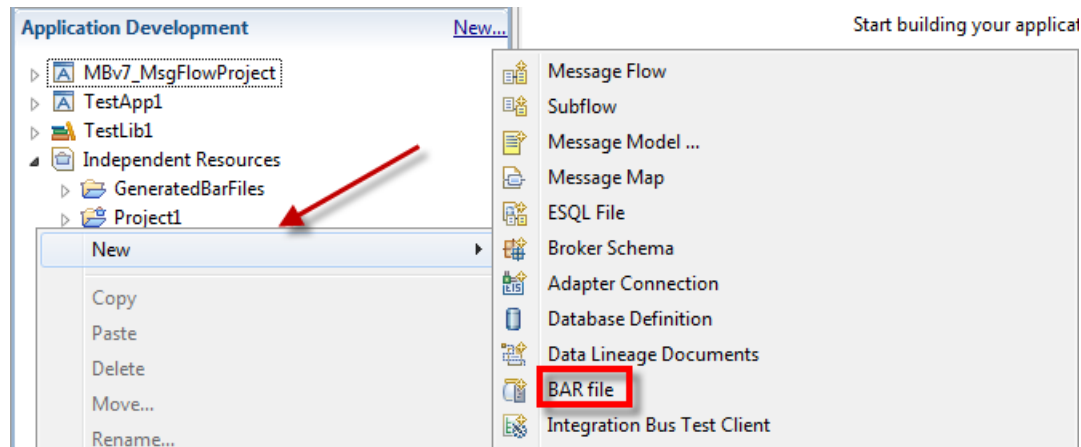
The new application should now be visible as an Application, and it is removed from the Independent Resources folder.



1.6 Bar file editor

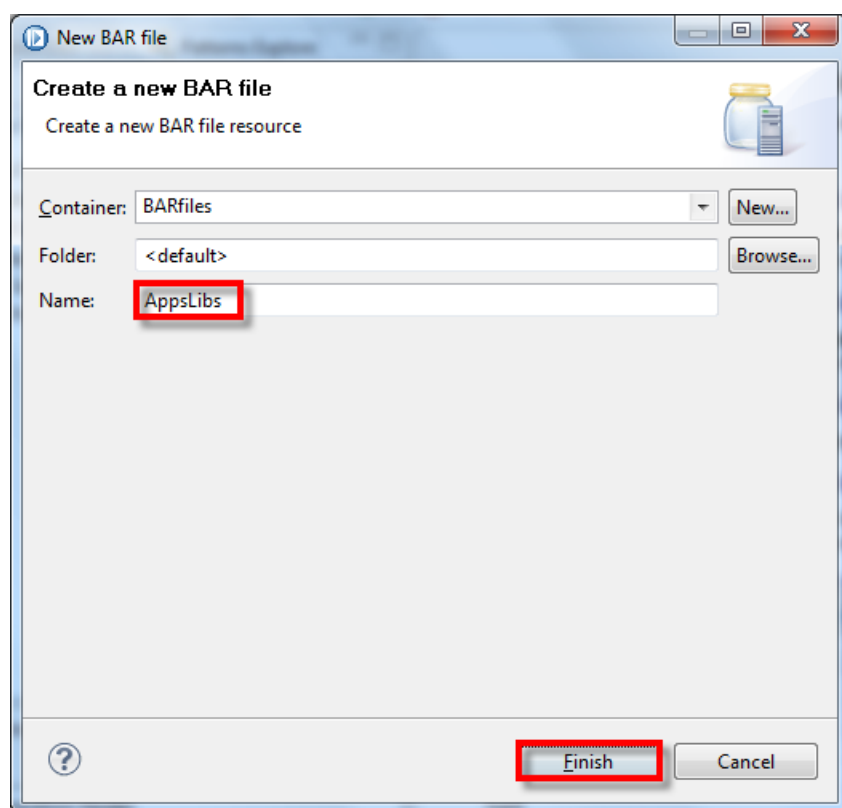
The bar file editor is updated to recognize applications and libraries. Deploy operations can be implemented on an application basis, or can continue to be utilized for individual projects.

- ___1. Point the mouse to a blank area of the project navigator.



- ___2. Press the right mouse button.
- ___3. Select **New→BAR file** from the menu.

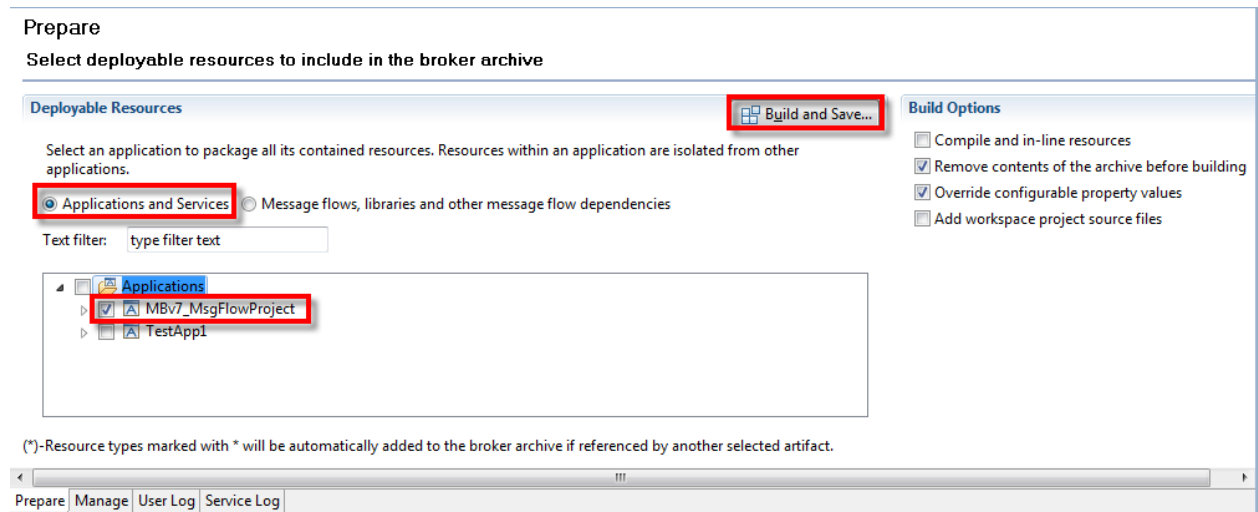
- __4. Enter **AppsLibs** as the **Name**.



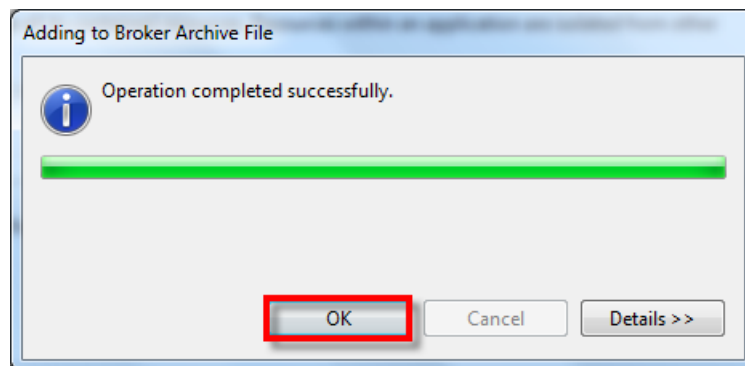
- __5. Press the **Finish** button to create the broker archive file.

The broker archive editor will open.

- __6. Confirm that the **Applications and Services** radio button is selected.



- __7. Select the **MBv7_MsgFlowProject** application.
- __8. Press the **Build and Save...** button.
- __9. Press the **OK** button to dismiss the pop-up dialog.



- ___10. Select the **Manage** tab to examine the contents of the broker archive file.

Manage

Rebuild, remove, edit, add resources to broker archive and configure their properties

Name	Type	Modified	Size	Path
MBv7_MsgFlowProject	Application	Jul 24, 2013 10:56:32 PM	3598	

Command for packaging the BAR contents

Prepare **Manage** User Log Service Log

- ___11. Select the **Prepare** tab.

Prepare

Select deployable resources to include in the broker archive

Deployable Resources

There are also resources selected from the 'Applications' view. Switch views to see these resources.

Applications and Services **Message flows, libraries and other message flow dependencies**

Text filter: type filter text Working set filter: <all resources>

Libraries*
TestLib1
Independent resources
Flows
Flow1.msgflow - /Project1/Flow1.msgflow

Build and Save...

Build Options

☐ Compile and in-line resources
☒ Remove contents of the archive before building
☒ Override configurable property values
☐ Add workspace project source files

(*)-Resource types marked with * will be automatically added to the broker archive if referenced by another selected artifact.

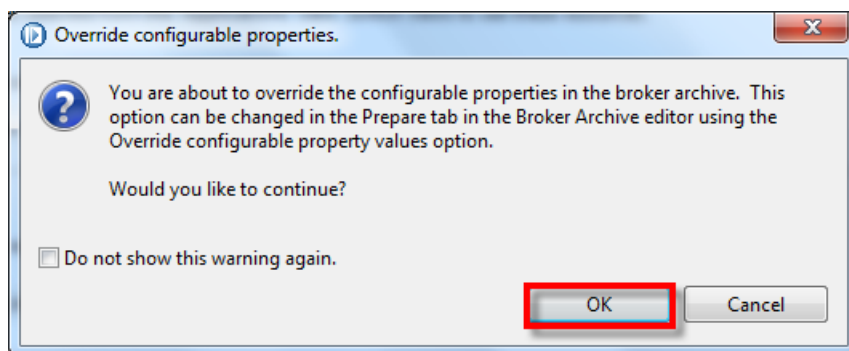
Prepare **Manage** User Log Service Log

- ___12. Select the **Message flows, libraries and other message flow dependencies** radio button.

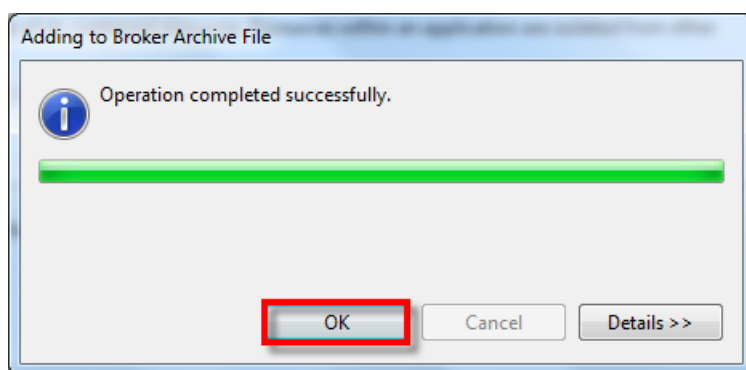
- ___13. Select the **Flow1.msgflow** message flow that was built earlier.

- ___14. Press the **Build and Save...** button.

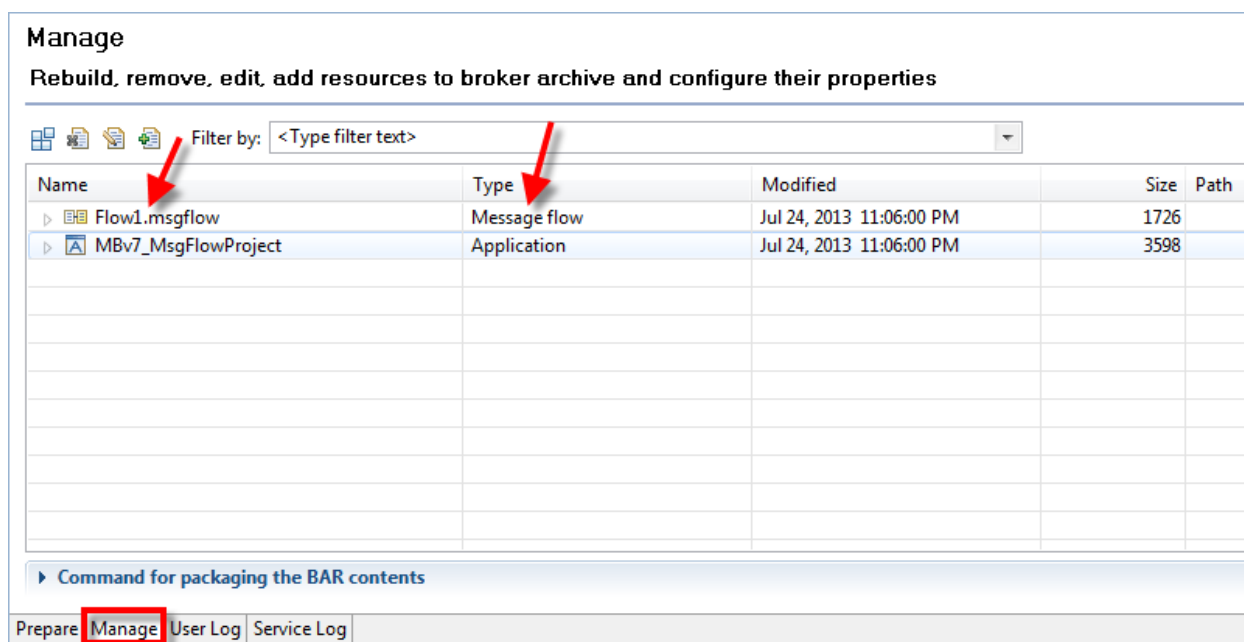
- ___15. If you get a pop-up dialog to Override configurable properties, click **OK**.



- ___16. Press the **OK** button to dismiss the pop-up dialog.

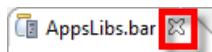


- ___17. Select the **Manage** tab.



The broker archive file now contains both an application and a message flow.

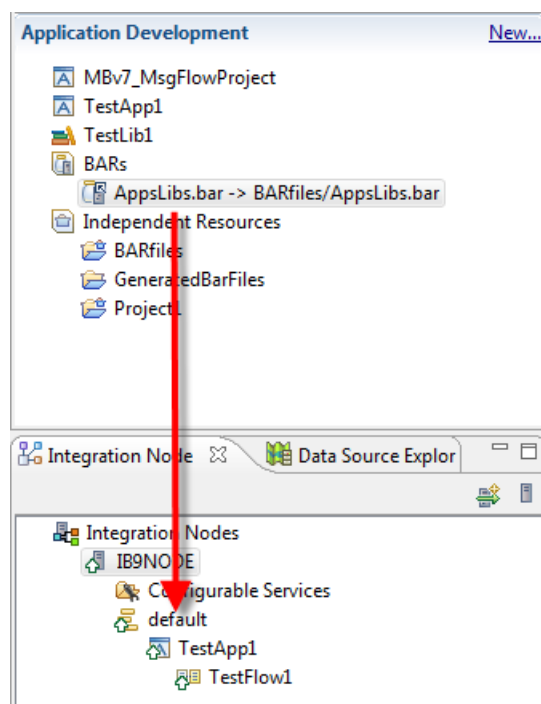
___18. Close the **AppsLibs.bar** file.



1.7 Deploying resources

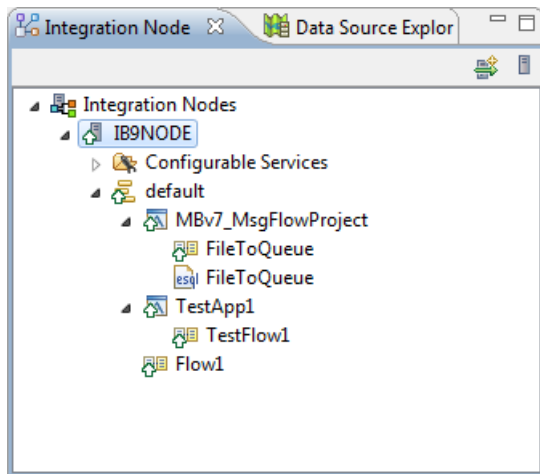
The broker archive file that was created in the previous section will now be deployed to the default execution group.

___1. Select the **AppsLibs.bar** broker archive file.



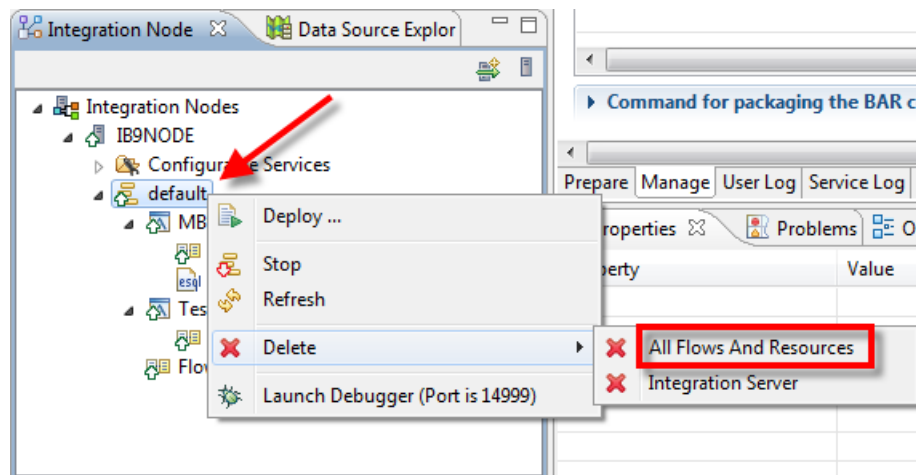
__2. Drag it to the **default** Integration Server and drop it.

At this point the **default** Integration Server contains two applications and an independent flow.



The lab is now complete. The deployed flows and other resources will now be removed from the Integration Server.

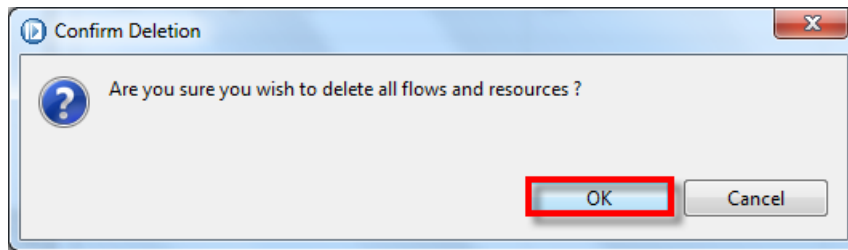
__3. Select the **default** Integration Server.



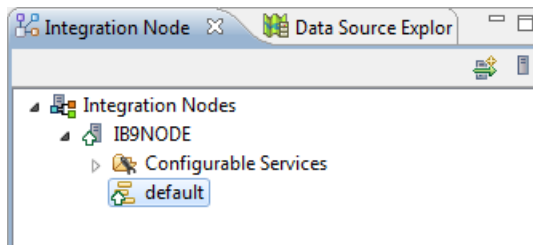
__4. Press the right mouse button.

__5. Select **Delete→All Flows and Resources** from the menu.

- __6. Press the **OK** button to confirm the deletion.



The execution group should now be empty.



1.8 Summary

Applications and libraries are a major enhancement in IBM Integration Bus V9. They have far reaching implications and should make managing the Integration Bus environment substantially easier.

This lab provides a basic introduction to applications and libraries.

This is the end of Lab 1.

Page Intentionally Left Blank

Lab 2 DFDL and Message Model tooling

2.1 Introduction to Message Model standards

A message model is used by IBM Integration Bus to model a message format. The message models are all based on World Wide Web Consortium (W3C) XML Schema 1.0 (XSD).

XML Schema is an international standard that defines a language for describing the structure of XML documents. It is suited to describing the messages that flow between business applications, and it is widely used in the business community for this purpose. IBM Integration Bus uses models that are based on XML Schema to describe the structure of all kinds of message formats, including message formats that are not XML.

Data Format Description Language 1.0 (DFDL) is an open standard modeling language from the Open Grid Forum (OGF) that builds upon the features of XML Schema 1.0 in order to model and validate all kinds of general text and binary data. It uses standard XSD model objects to describe the logical structure of data, together with DFDL annotations that describe the physical text or binary representation of data. IBM Integration Bus uses DFDL schema files to describe text and binary data, including industry standard formats.

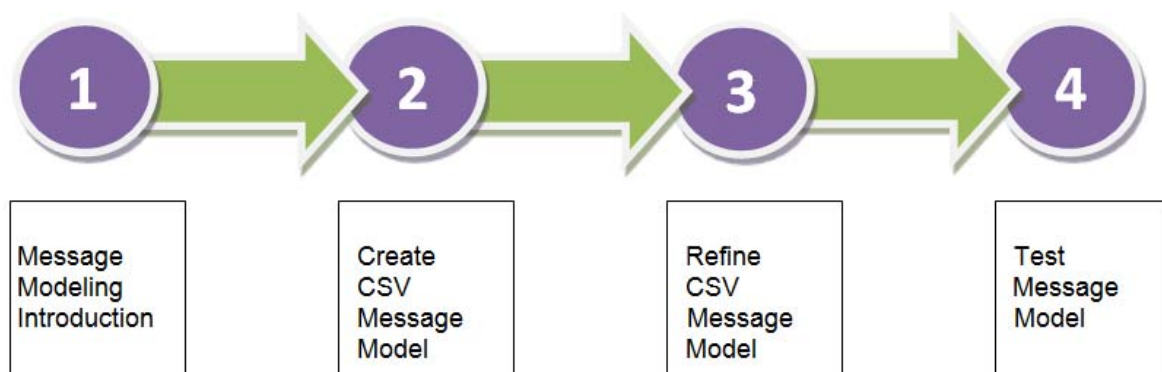
DFDL is not intended to be used to model XML documents (use normal XML Schema files).

Support for DFDL in IBM Integration Bus includes:

- DFDL parser and domain
- DFDL schema file creation wizards
- DFDL schema editor for modeling text and binary data formats
- DFDL Test perspective for testing your DFDL schema files

For more information about DFDL, you can go to the [Open Grid Forum \(OGF\)](#) web site. IBM Integration Bus supports DFDL 1.0, as defined in the following document: [Data Format Description Language \(DFDL\) 1.0](#).

Unlike previous versions of IBM Integration Bus, you do not need to create a Message Set project and a Message Set to model a message type (although MRM Message Models are still supported); you just have to create a DFDL schema file.



2.2 Creating a CSV Message Model

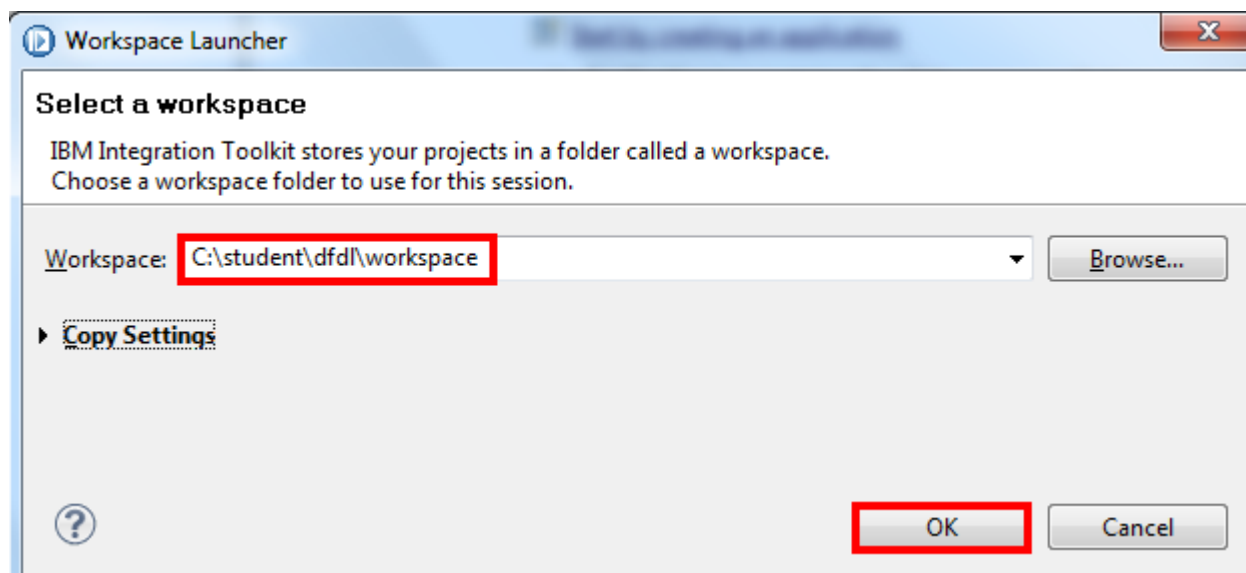
This lab will create a Message Model that will model a CSV file. The file has a header record, and several detail records, but no trailer record.

Records are delimited by CRLF characters, and fields within each record are delimited by a comma.

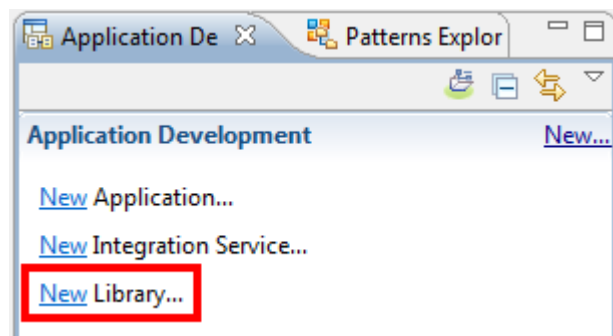
The Library project type will be used to store the Message Model.

This lab will start from a blank workspace.

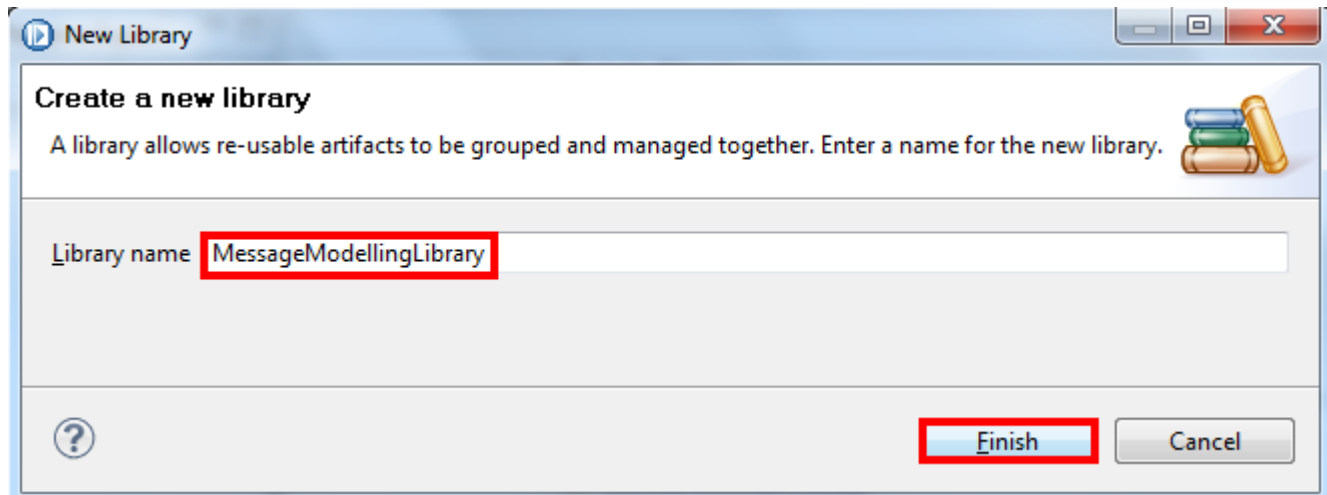
- __1. Select **File→Switch Workspace→Other**.
- __2. Select the **C:\student\DFDL\workspace** workspace.
- __3. Press the **OK** button to continue.



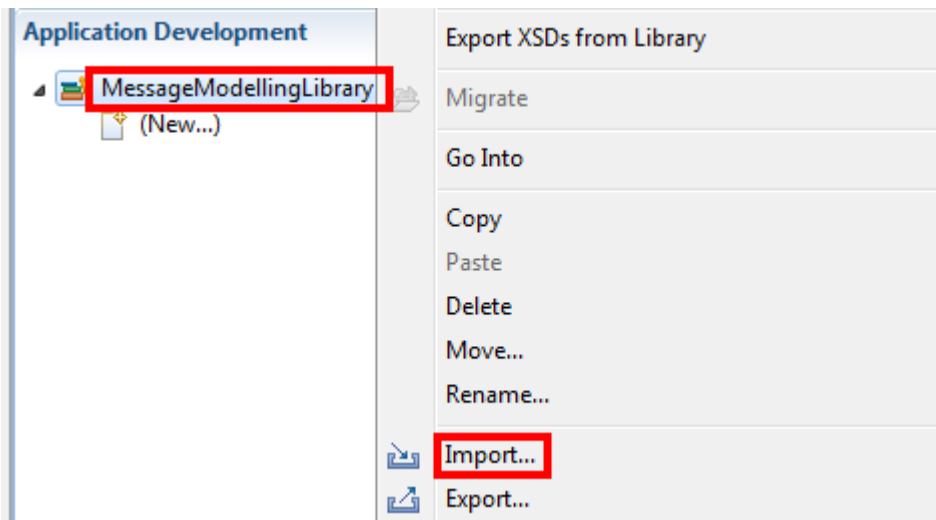
- __4. Click on **New Library**.



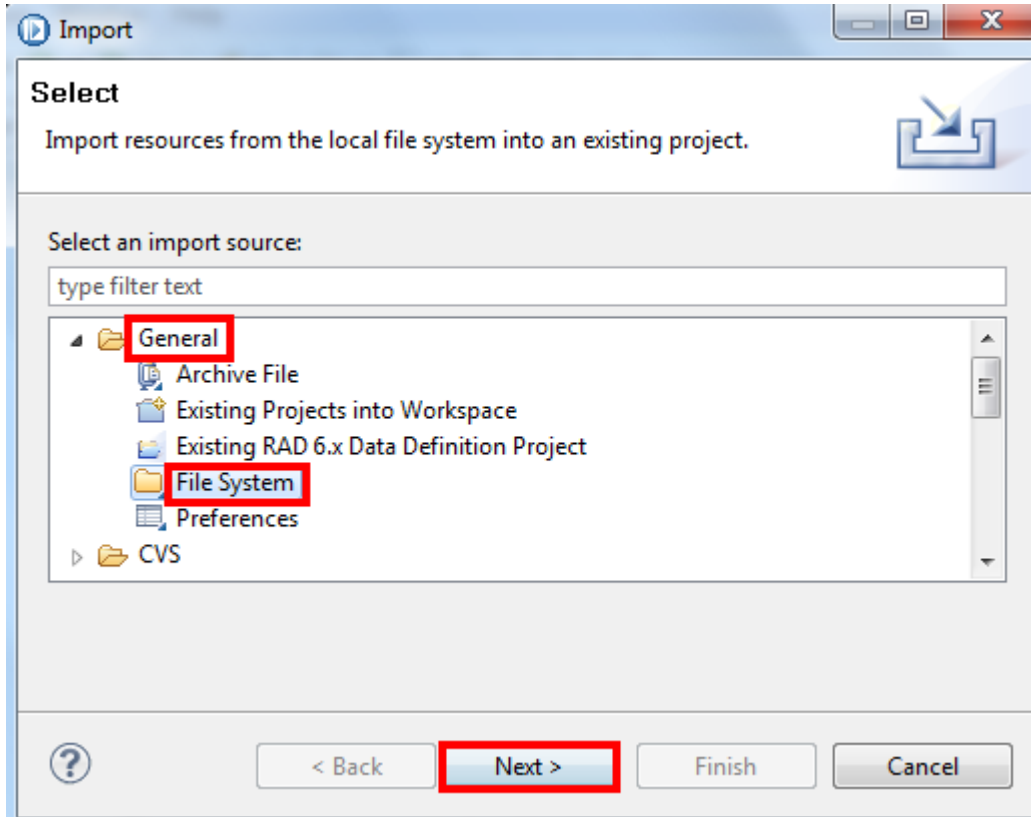
- __5. Set the **Library name** to **MessageModellingLibrary**.
- __6. Press the **Finish** button.



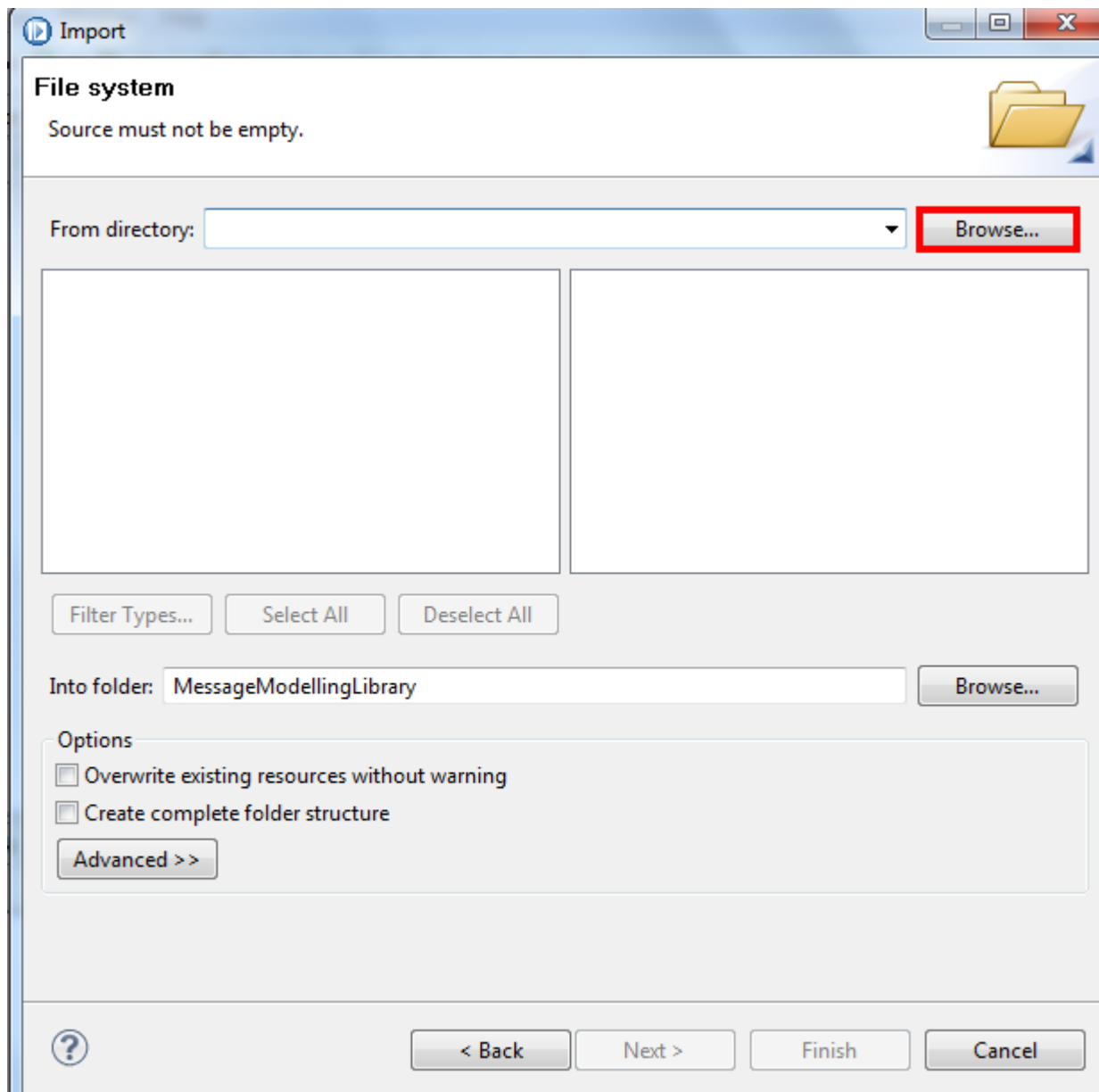
- __7. Select the **MessageModellingLibrary** library.
- __8. Press the right mouse button.
- __9. Select **Import** from the menu



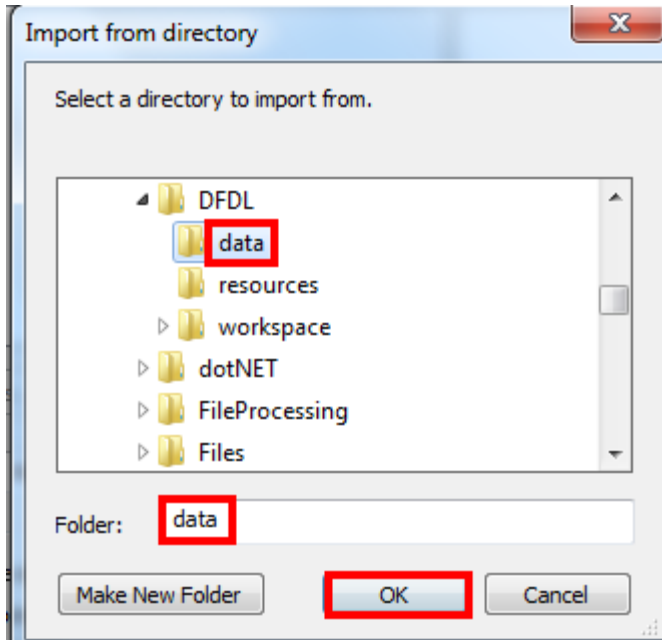
- __10. Expand the **General** folder.
- __11. Select **File System**.
- __12. Press the **Next** button.



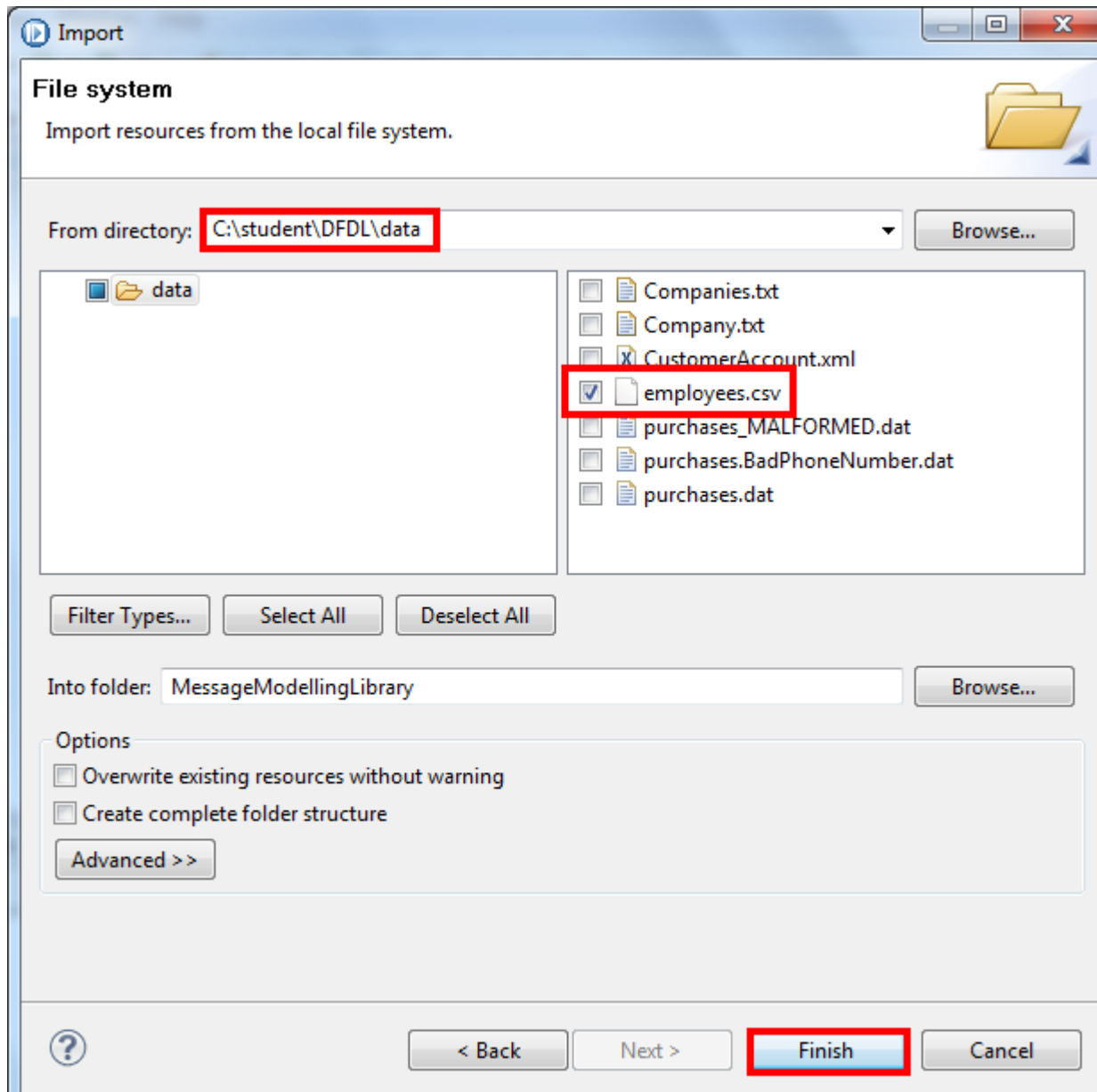
___13. Press the **Browse** button for **From directory**.



- __14. Navigate to the **C:\student\DFDL\data** directory.
- __15. Select the **data** directory.
- __16. Press the **OK** button.

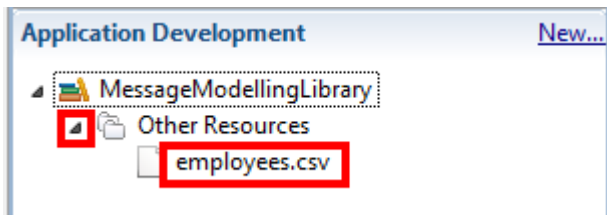


- __17. Select the **employees.csv** file.
- __18. Press the **Finish** button to perform the import.



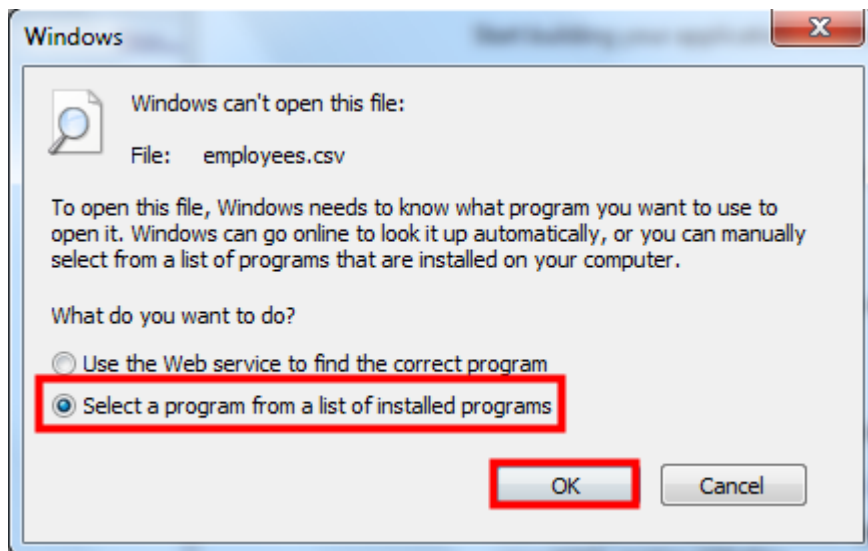
__19. Expand the **Other Resources** folder.

__20. Double-click the **employees.csv** file.

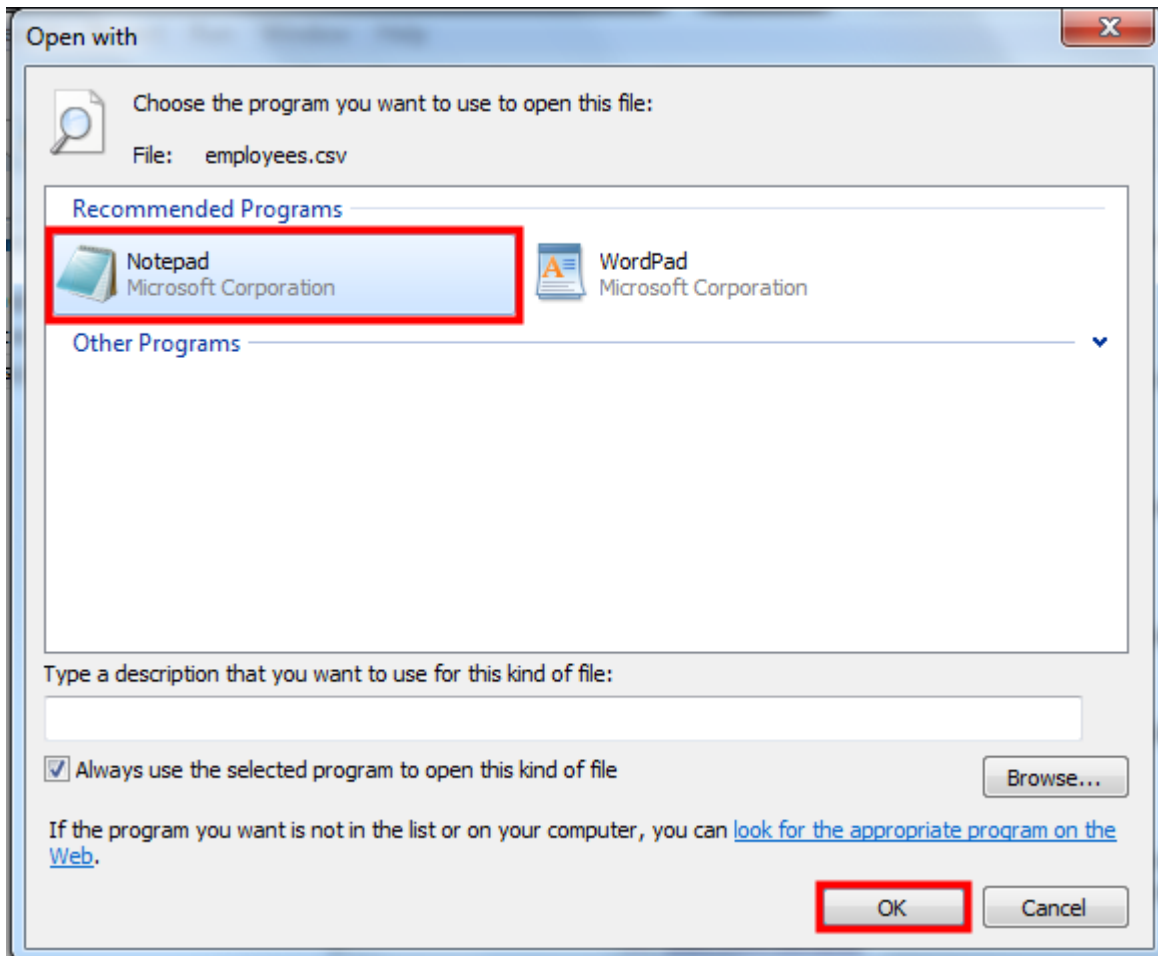


__21. Select the **Select a program from a list of installed programs** radio button.

__22. Press the **OK** button.



- __23. Select the **Notepad** utility.
- __24. Press the **OK** button.



The file data will be opened in a notepad session. There is a header record and 42 detail records. Each detail record has 12 fields.

__25. Examine the file data.

__26. Close the notepad window.

```

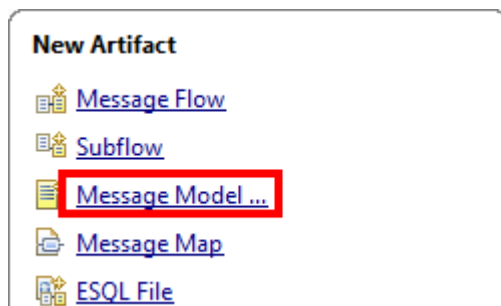
Company1,42.123 Street,Winchester,UK
"000010","CHRISTINE","I","HAAS","A00","3978","1995-01-01","PRES","18","F","1963-08-24",152750.00
"000020","MICHAEL","L","THOMPSON","B01","3476","2003-10-10","MANAGER","18","M","1978-02-02",94250.00
"000030","SALLY","A","KWAN","C01","4738","2005-04-05","MANAGER","20","F","1971-05-11",98250.00
"000050","JOHN","B","GEYER","E01","6789","1979-08-17","MANAGER","16","M","1955-09-15",80175.00
"000060","IRVING","F","STERN","D11","6423","2003-09-14","MANAGER","16","M","1975-07-07",72250.00
"000070","EVA","D","PULASKI","D21","7831","2005-09-30","MANAGER","16","F","2003-05-26",96170.00
"000090","EILEEN","W","HENDERSON","E11","5498","2000-08-15","MANAGER","16","F","1971-05-15",89750.00
"000100","THEODORE","Q","SPENSER","E21","0972","2000-06-19","MANAGER","14","M","1980-12-18",86150.00
"000110","VINCENTO","G","LUCCHESI","A00","3490","1988-05-16","SALESREP","19","M","1959-11-05",66500.00
"000120","SEAN","O","CONNELL","A00","2167","1993-12-05","CLERK","14","M","1972-10-18",49250.00
"000130","DELORES","M","QUINTANA","C01","4578","2001-07-28","ANALYST","16","F","1955-09-15",73800.00
"000140","HEATHER","A","NICHOLLS","C01","1793","2006-12-15","ANALYST","18","F","1976-01-19",68420.00
"000150","BRUCE","A","ADAMSON","D11","4510","2002-02-12","DESIGNER","16","M","1977-05-17",55280.00
"000160","ELIZABETH","R","PIANKA","D11","3782","2006-10-11","DESIGNER","17","F","1980-04-12",62250.00
"000170","MASATOSHI","J","YOSHIMURA","D11","2890","1999-09-15","DESIGNER","16","M","1981-01-05",44680.00
"000180","MARILYN","S","SCOUTTEN","D11","1682","2003-07-07","DESIGNER","17","F","1979-02-21",51340.00
"000190","JAMES","M","WALKER","D11","2986","2004-07-26","DESIGNER","16","M","1982-06-25",50450.00
"000200","DAVID","B","BROWN","D11","4501","2002-03-03","DESIGNER","16","M","1971-05-29",57740.00
"000210","WILLIAM","T","JONES","D11","0942","1998-04-11","DESIGNER","17","M","2003-02-23",68270.00
"000220","JENNIFER","K","LUTZ","D11","0672","1998-08-29","DESIGNER","18","F","1978-03-19",49840.00
"000230","JAMES","J","JEFFERSON","D21","2094","1996-11-21","CLERK","14","M","1980-05-30",42180.00
"000240","SALVATORE","M","MARINO","D21","3780","2004-12-05","CLERK","17","M","2002-03-31",48760.00
"000250","DANIEL","S","SMITH","D21","0961","1999-10-30","CLERK","15","M","1969-11-12",49180.00
"000260","SYBIL","P","JOHNSON","D21","8953","2005-09-11","CLERK","16","F","1976-10-05",47250.00
"000270","MARIA","L","PEREZ","D21","9001","2006-09-30","CLERK","15","F","2003-05-26",37380.00
"000280","ETHEL","R","SCHNEIDER","E11","8997","1997-03-24","OPERATOR","17","F","1976-03-28",36250.00
"000290","JOHN","R","PARKER","E11","4502","2006-05-30","OPERATOR","12","M","1985-07-09",35340.00
"000300","PHILIP","X","SMITH","E11","2095","2002-06-19","OPERATOR","14","M","1976-10-27",37750.00
"000310","MAUDE","F","SETRIGHT","E11","3332","1994-09-12","OPERATOR","12","F","1961-04-21",35900.00
"000320","RAMLAL","V","MEHTA","E21","9990","1995-07-07","FIELDREP","16","M","1962-08-11",39950.00
"000330","WING","L","LEE","E21","2103","2006-02-23","FIELDREP","14","M","1971-07-18",45370.00
"000340","JASON","R","GOUNOT","E21","5698","1977-05-05","FIELDREP","16","M","1956-05-12",43840.00
"000010","DIAN","J","HEMMINGER","A00","3978","1995-01-01","SALESREP","18","F","1973-08-14",46500.00
"200120","GREG","A","ORLANDO","A00","2167","2002-05-05","CLERK","14","M","1972-10-18",39250.00
"200140","KIM","N","NATZ","C01","1793","2006-12-15","ANALYST","18","F","1976-01-19",68420.00

```

__27. In the Integration Toolkit click on **New**.



__28. Select **Message Model** from the menu.



__29. Select the **CSV text** radio button.

__30. Press the **Next** button.

New Message Model

Create a new message model file

Select the message model type or format

XML

- ☐ **SOAP XML** XML data for use in Web Services.
- ☐ **Other XML** All other XML data.

Text and binary

- ☒ **CSV text** Comma Separated Values data, a delimited text format commonly used as an export format by spreadsheets and databases.
- ☐ **Record-oriented text** Text data formats where delimited fields are grouped into records.
- ☐ **COBOL** Data for COBOL programs
- ☐ **C** Data for C programs
- ☐ **Other text or binary** All other text or binary data formats.

Enterprise Information Systems

- ☐ **SAP** Data from SAP systems including IDoc and BAPI
- ☐ **Siebel** Data from Siebel systems
- ☐ **PeopleSoft** Data from PeopleSoft
- ☐ **JD Edwards** Data from JD Edwards systems

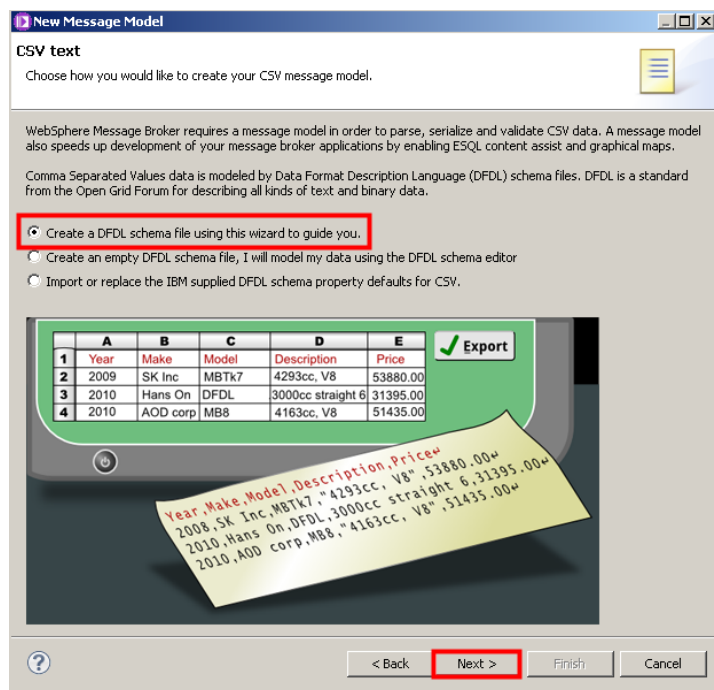
Other

- ☐ **CORBA IDL** Data from CORBA
- ☐ **Database record** Records from relational databases
- ☐ **MIME** Data for extended email format
- ☐ **IBM supplied** Predefined data format

Navigation:

-
-
-
-
-

- ___31. Select the **Create a DFDL schema file using the wizard to guide you** radio button.
- ___32. Press the **Next** button.



The dialog box is titled "New Message Model" and has a "CSV text" subtitle. It contains instructions about WebSphere Message Broker requirements and DFDL schema files. Three radio buttons are present, with the first one selected and highlighted by a red rectangle. Below the buttons is a preview of a CSV table and a yellow sticky note showing the corresponding DFDL schema. At the bottom, there are four buttons: "< Back", "Next >" (highlighted with a red rectangle), "Finish", and "Cancel".

Choose how you would like to create your CSV message model.

WebSphere Message Broker requires a message model in order to parse, serialize and validate CSV data. A message model also speeds up development of your message broker applications by enabling ESQL content assist and graphical maps.

Comma Separated Values data is modeled by Data Format Description Language (DFDL) schema files. DFDL is a standard from the Open Grid Forum for describing all kinds of text and binary data.

☒ Create a DFDL schema file using this wizard to guide you.

☐ Create an empty DFDL schema file, I will model my data using the DFDL schema editor

☐ Import or replace the IBM supplied DFDL schema property defaults for CSV.

	A	B	C	D	E
1	Year	Make	Model	Description	Price
2	2009	SK Inc	MBTk7	4293cc, V8	53880.00
3	2010	Hans On	DFDL	3000cc straight 6	31395.00
4	2010	AOD corp	MB8	4163cc, V8	51435.00

Export

Year,Make,Model,Description,Price
2009,SK Inc,MBTk7,"4293cc, V8",53880.00
2010,Hans On,DFDL,3000cc straight 6,31395.00
2010,AOD corp,MB8,"4163cc, V8",51435.00

< Back Next > Finish Cancel

- __33. Make sure that the **Project** is set to **MessageModellingLibrary**.
- __34. Enter **Employee** as the **DFDL schema file name**.
- __35. Press the **Next** button.

New Message Model

Create a Data Format Description Language (DFDL) Schema

Specify the location and name of the DFDL schema, and specify the name of the message.

Application or Library: MessageModellingLibrary Browse... New...

Folder: Browse... New...

DFDL schema file name: Employee

Message name: Employee

< Back Next > Finish Cancel

__36. Select the **first record is a header** check box.

__37. Enter **12** as the **Number of fields**.

__38. Press the **Finish** button.

New Message Model

Configure schema for CSV data
Provide settings for a new schema that will model CSV data.

Record settings
End of record character: Carriage Return & Line Feed - %CR;%LF;
(Blank records will be skipped)
☒ The first record is a header

Field settings
Number of fields: 12
☐ Create default values for fields

Encoding code page options:
☒ Dynamic (provided to the processor by the application at runtime)
☐ Fixed UTF-8

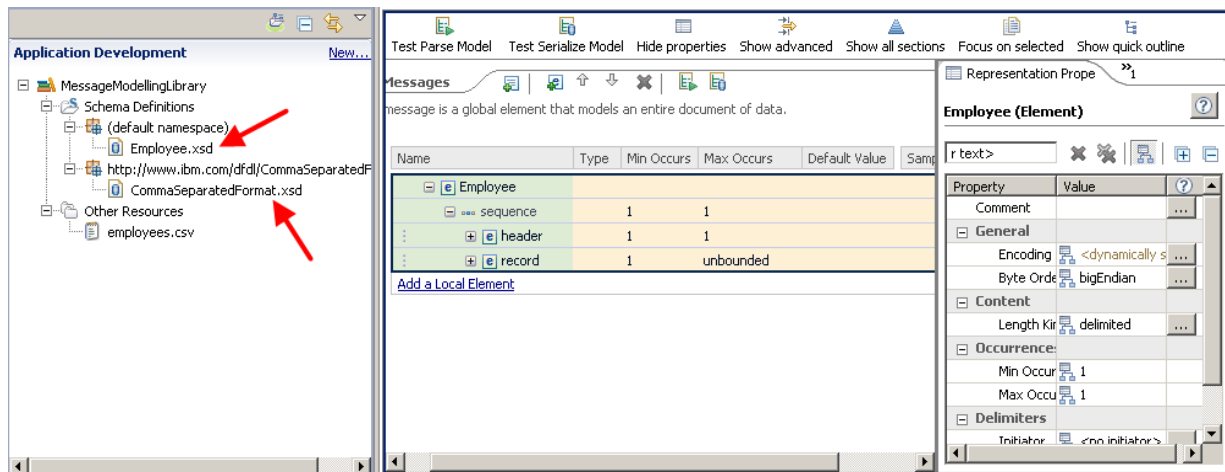
Global settings
Escape scheme: CSV Escape Scheme

Buttons: < Back, Next >, **Finish**, Cancel

There should be a new folder in the library with an XSD file called **Employee.xsd** (under the **default namespace**). This is the Message Model that was just created. This is a standard DFDL XSD, with no specific annotations for IBM Integration Bus.

There is another XSD file under Schema Definitions called "CommaSeparatedFormat.xsd". This "Helper schema file" was automatically added by the CSV wizard and contains CSV-specific defaults for all the DFDL properties. This is required because DFDL does not have built-in defaults, so if an object needs a property, a value must be supplied. To ease this task, the wizard creates a helper DFDL schema for each kind of data (such as COBOL and CSV.).

These files are related by an import statement in the schema references section of the **Employee.xsd** file.



Take a look at the Representation Properties view on the right hand side. Notice the "inheritance" icon to the right of the "**Length Kind**" property. (**Hint:** You might have to adjust the column lengths in the Representation Properties view to see the full contents.)

In this case, the "**Length Kind**" property was inherited from the "**CommaSeparatedFormat**" helper schema file that the wizard automatically imported.

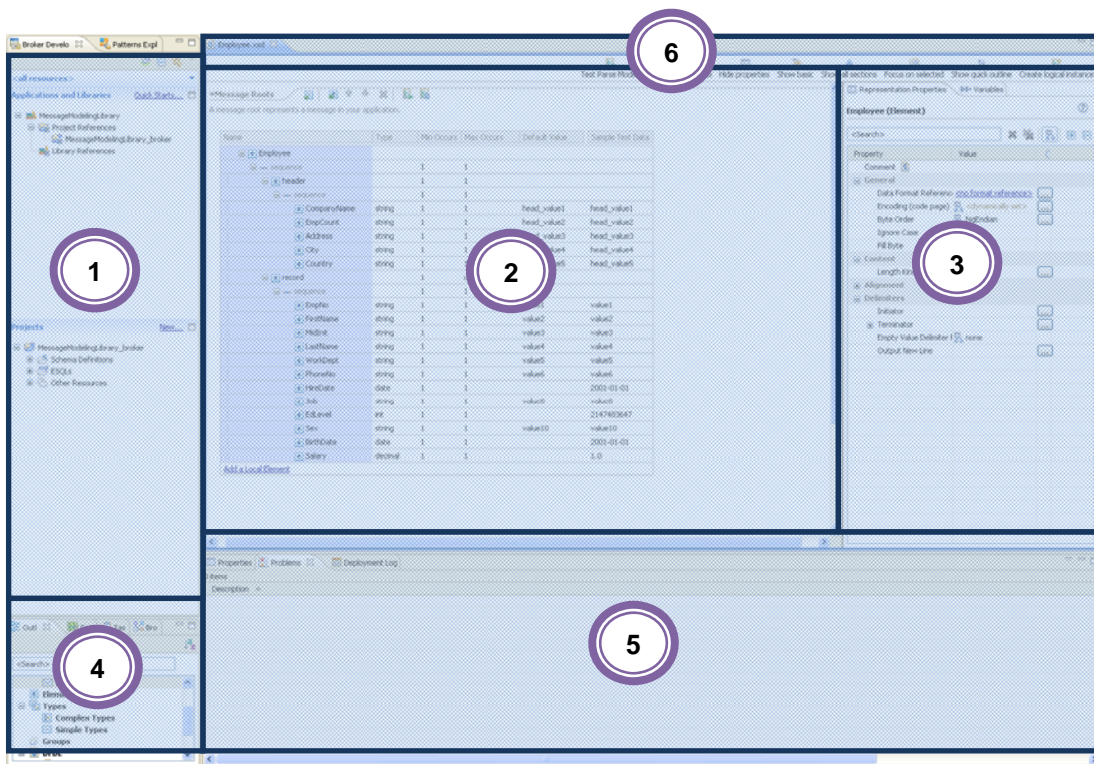
__39. Hover over the inheritance icon to discover its origin.

The screenshot shows the IBM Integration Bus v9.0 Representation Properties view for an **Employee (Element)**. The view is divided into two main sections: a table of properties on the left and a detailed view of the selected property on the right.

The table of properties has the following columns: Name, Type, Min Occurs, Max Occurs, Default Value, and Sample Value. The data is as follows:

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		

The right-hand section shows the **Representation Properties** for the **Employee (Element)**. It includes a search bar, a table of properties, and a detailed view of the selected property. The **Length Kind** property is highlighted in yellow, and a red arrow points to the inheritance icon. A tooltip is displayed over the inheritance icon, stating: "Property inherited from global named format {http://www.ibm.com/dfdl/CommaSeparatedFormat}CommaSeparatedFormat".



Take a look at the different parts of the DFDL Editor:

- 1) **Applications and Libraries / Projects view:** In the top of this view you will find the Applications/Libraries and in the bottom the projects. All the DFDL XSD files will be located under the "Schema Files" folder inside of a project.
- 2) **Logical structure view:** This is the core of the DFDL Editor. Here you will work with the Message Models.
- 3) **DFDL properties:** Shows the DFDL properties of the selected elements in the editor view. These were moved from the traditional properties window to provide better viewing.
- 4) **Outline view:** Shows the complete outline of the current DFDL Message Model.
- 5) **Problems view:** In this view you will be able to see all the warning and errors of your project.
- 6) **Icon Bar:** Several actions can be started from the icons in this bar:



- **Test Parse Model:** Launches the DFDL Test Perspective to test-parse sample data against your selected message model.
- **Test Serialize Model:** Launches the DFDL Test Perspective, to test-serialize sample data by using the selected DFDL schema.
- **Hide properties:** Hides the Representation Properties view.
- **Show advanced:** Shows the advanced Representation Properties.
- **Show all sections:** Shows all the available sections of the DFDL Message Model in the Editor.
- **Focus on selected:** Shows only the selected element in the Editor.
- **Show quick outline:** Shows a hanging outline view of the message model in the Editor.

The DFDL Editor in the center of the screen describes data elements. Expand the header element.

Take a look at the columns:

- Name: name of the data element
- Type: data type (such as string, int, Boolean, decimal or date)
- Min Occurs: minimum amount of occurrences expected (0 or greater)
- Max Occurs: maximum amount of occurrences expected (from 1 to unbounded)
- Default Value: used to provide the logical value of a required element while parsing or serializing messages when the element is missing
- Sample Test Data: can be used to generate a logical instance of the model as you will see later on

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
[-] [e] Employee					
[-] ... sequence		1	1		
[-] [e] header		1	1		
[-] ... sequence		1	1		
[e] head_field1	string	1	1		head_value1
[e] head_field2	string	1	1		head_value2
[e] head_field3	string	1	1		head_value3
[e] head_field4	string	1	1		head_value4
[e] head_field5	string	1	1		head_value5
[e] head_field6	string	1	1		head_value6
[e] head_field7	string	1	1		head_value7
[e] head_field8	string	1	1		head_value8
[e] head_field9	string	1	1		head_value9
[e] head_field10	string	1	1		head_value10
[e] head_field11	string	1	1		head_value11
[e] head_field12	string	1	1		head_value12
[+] [e] record		1	unbounded		
Add a Local Element					

Fields 6 to 12 will now be deleted from the **header** element.

__40. Expand the **header** element.

__41. Select **head_field6**.

__42. Hold down the **Ctrl** key and select fields **head_field7** through **head_field12**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
... sequence		1	1		
...					
header		1	1		
... sequence		1	1		
...					
head_field1	string	1	1		head_value1
head_field2	string	1	1		head_value2
head_field3	string	1	1		head_value3
head_field4	string	1	1		head_value4
head_field5	string	1	1		head_value5
head_field6	string	1	1		head_value6
head_field7	string	1	1		head_value7
head_field8	string	1	1		head_value8
head_field9	string	1	1		head_value9
head_field10	string	1	1		head_value10
head_field11	string	1	1		head_value11
head_field12	string	1	1		head_value12
...					
record		1	unbounded		
Add a Local Element					

__43. Press the right mouse button.

__44. Select **Delete**.

N.B. You can also just press the **Delete** key.

...				1	head_value6
...				1	head_value7
...				1	head_value8
...				1	head_value9
head_field10	string	1	1		head_value10
head_field11	string	1	1		head_value11
head_field12	string	1	1		head_value12
...					
record		1	unbounded		
Add a Local Element					

Change the names of the remaining five header fields as shown.

__45. Click on the name of the first element.

__46. Change the name as shown.

__47. Use the down arrow key to move to the next element name. Press the **Enter** key when the last name is changed.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
sequence		1	1		
CompanyName	string	1	1		head_value1
EmpCount	string	1	1		head_value2
Address	string	1	1		head_value3
City	string	1	1		head_value4
Country	string	1	1		head_value5
record		1	unbounded		

[Add a Local Element](#)

__48. Collapse the **header** element.

__49. Expand the **record** element to show the 12 fields created by the wizard.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
... sequence		1	1		
⋮ header		1	1		
⋮ record		1	unbounded		
... sequence		1	1		
⋮ field1	string	1	1		value1
⋮ field2	string	1	1		value2
⋮ field3	string	1	1		value3
⋮ field4	string	1	1		value4
⋮ field5	string	1	1		value5
⋮ field6	string	1	1		value6
⋮ field7	string	1	1		value7
⋮ field8	string	1	1		value8
⋮ field9	string	1	1		value9
⋮ field10	string	1	1		value10
⋮ field11	string	1	1		value11
⋮ field12	string	1	1		value12
Add a Local Element					

__50. Change the field names to the following:

- | | |
|--------------|---------------|
| 1) EmpNo | 7) HireDate |
| 2) FirstName | 8) Job |
| 3) MidInit | 9) EdLevel |
| 4) LastName | 10) Sex |
| 5) WorkDept | 11) BirthDate |
| 6) PhoneNo | 12) Salary |

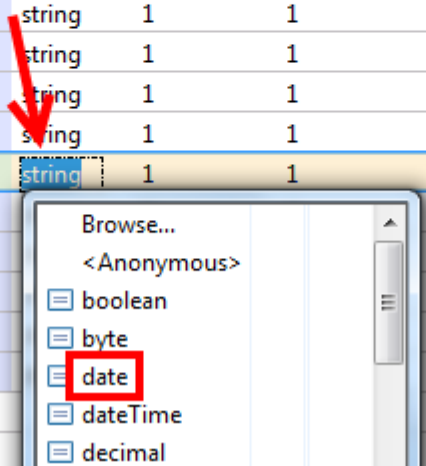
Again use the down arrow to move from one field to the next. Press the **Enter** key to complete the rename operation.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo	string	1	1		value1
FirstName	string	1	1		value2
MidInit	string	1	1		value3
LastName	string	1	1		value4
WorkDept	string	1	1		value5
PhoneNo	string	1	1		value6
HireDate	string	1	1		value7
Job	string	1	1		value8
EdLevel	string	1	1		value9
Sex	string	1	1		value10
BirthDate	string	1	1		value11
Salary	string	1	1		value12
Add a Local Element					

__51. Click in the **Type** column for the **HireDate** field.

__52. Select **date** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo	string	1	1		value1
FirstName	string	1	1		value2
MidInit	string	1	1		value3
LastName	string	1	1		value4
WorkDept	string	1	1		value5
PhoneNo	string	1	1		value6
HireDate	string	1	1		value7
Job					value8
EdLevel					value9
Sex					value10
BirthDate					value11
Salary					value12
Add a Local Element					



__53. Click in the **Type** column for the **BirthDate** field.

__54. Select **date** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo					value1
FirstName					value2
MidInit					value3
LastName					value4
WorkDept					value5
PhoneNo					value6
HireDate					2010-12-31
Job					value8
EdLevel					value9
Sex					value10
BirthDate	string	1	1		value11
Salary	string	1	1		value12
Add a Local Element					

Browse...

<Anonymous>

- boolean
- byte
- date**
- dateTime
- decimal
- double
- float
- hexBinary

__55. Click in the **Type** column for the **EdLevel** field.

__56. Select **int** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
Employee					
sequence		1	1		
header		1	1		
record		1	unbounded		
sequence		1	1		
EmpNo	string	1	1		value1
FirstName	string	1	1		value2
MidInit	string	1	1		value3
LastName	string	1	1		value4
WorkDept	string	1	1		value5
PhoneNo	string	1	1		value6
HireDate	date	1	1		2010-12-31
Job	string	1	1		value8
EdLevel	string	1	1		value9
Sex					value10
BirthDate					2010-12-31
Salary					value12
Add a Local Element					

boolean
byte
date
dateTime
decimal
double
float
hexBinary
int
integer

__57. Click in the **Type** column for the **Salary** field.

__58. Select **decimal** as the **Type**.

Name	Type	Min Occurs	Max Occurs	Default Value	Sample Value
[-] Employee					
[-] sequence		1	1		
[+] header		1	1		
[-] record		1	unbounded		
[-] sequence		1	1		
EmpNo	string	1	1		value1
FirstName					value2
MidInit					value3
LastName					value4
WorkDept					value5
PhoneNo					value6
HireDate					2010-12-31
Job					value8
EdLevel					1
Sex					value10
BirthDate					2010-12-31
Salary	string	1	1		value12

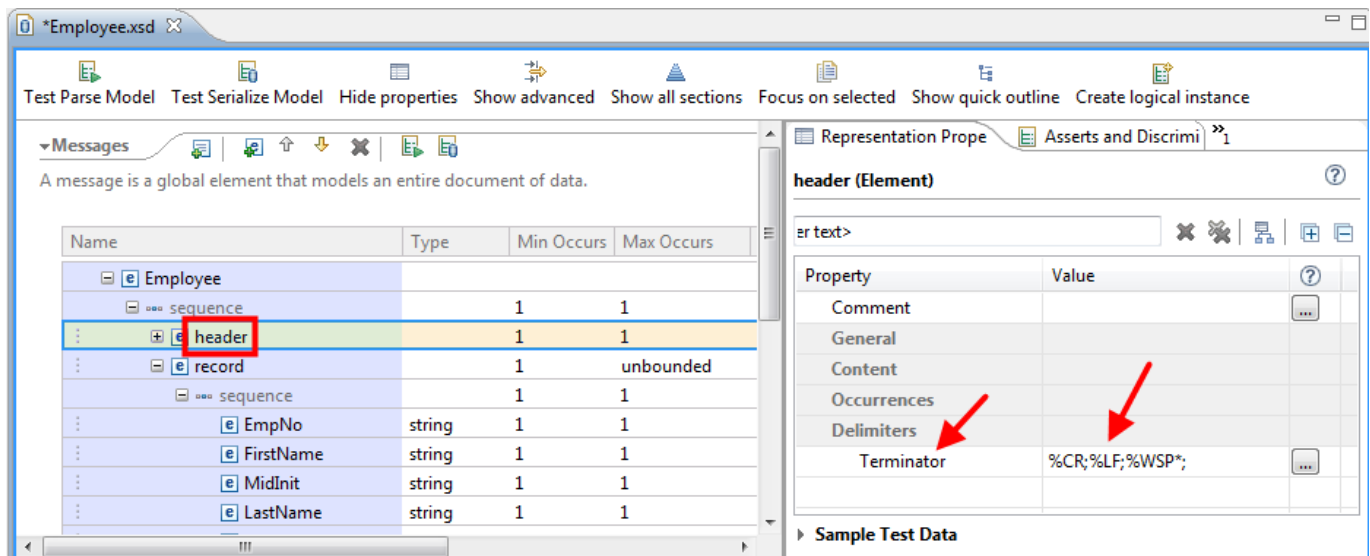
Browse...

<Anonymous>

- boolean
- byte
- date
- dateTime
- decimal**
- double
- float
- hexBinary

[Add a Local Element](#)

59. Select the **header** field under the **Employee** → **sequence** elements.



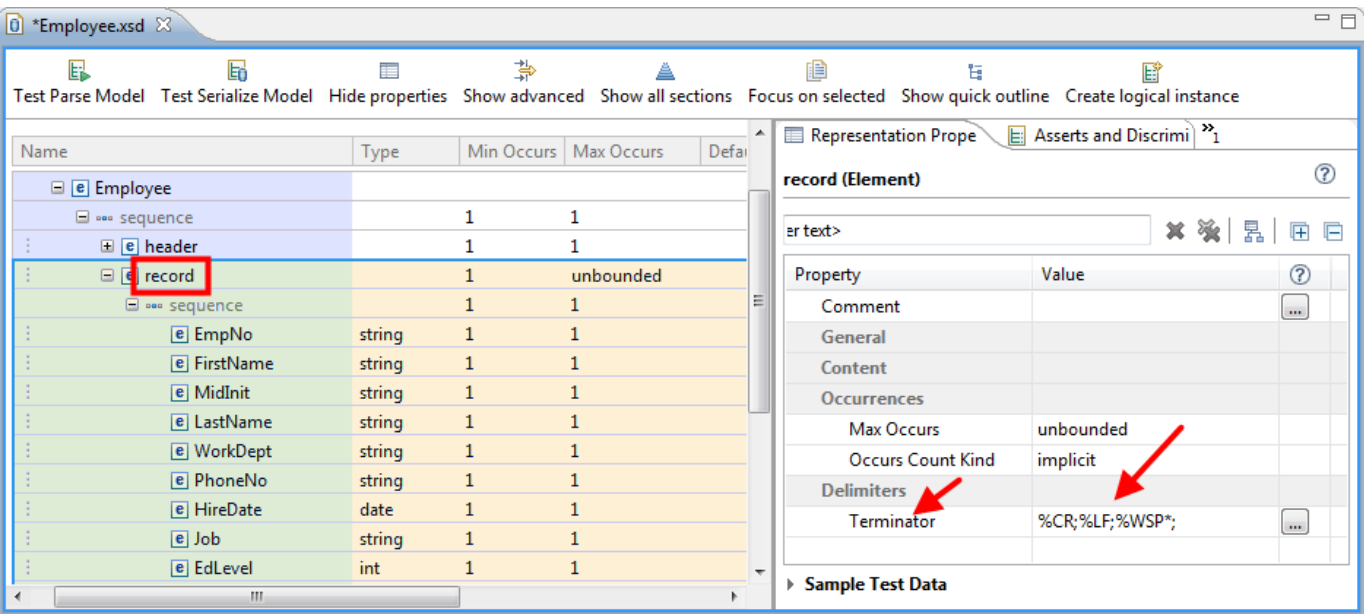
The terminator is the delimiter between records or between the header and the records. Since a CRLF sequence does not consist of printable characters the terminator is entered using special character sequences as shown.

There is a third Separator "%WSP*" which was added by the wizard because it automatically skips blank records (white space). "%WSP" (whitespace) is a DFDL character entity which groups several whitespace characters.

"%WSP*" (optional whitespaces) is a DFDL character class which implies the parser will ignore whitespace characters.

__60. Select the **record** element (if necessary).

__61. Examine the **Terminator** property.



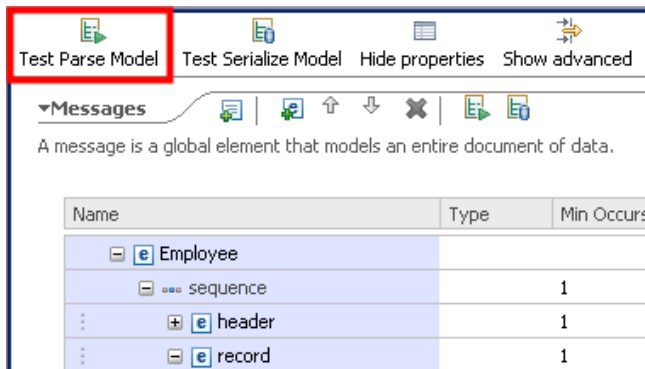
__62. Save the Message Model (**Ctrl+S**).



2.3 Test the Message Model

The Message Model will be tested to validate that it parses the sample data correctly.

___1. Click on the **Test Parse Model** button.



- __2. Select the **Content from a data file** radio button.
- __3. Press the **Browse** button.

Test Parse Model

Message
Select message for testing. [More...](#)
Message name:* Employee

Parser Input
Select content to be parsed against schema.
☐ Content from 'DFDL Test - Serialize' view
☒ Content from a data file
Input file name:* **Browse...**

Specify runtime configuration.

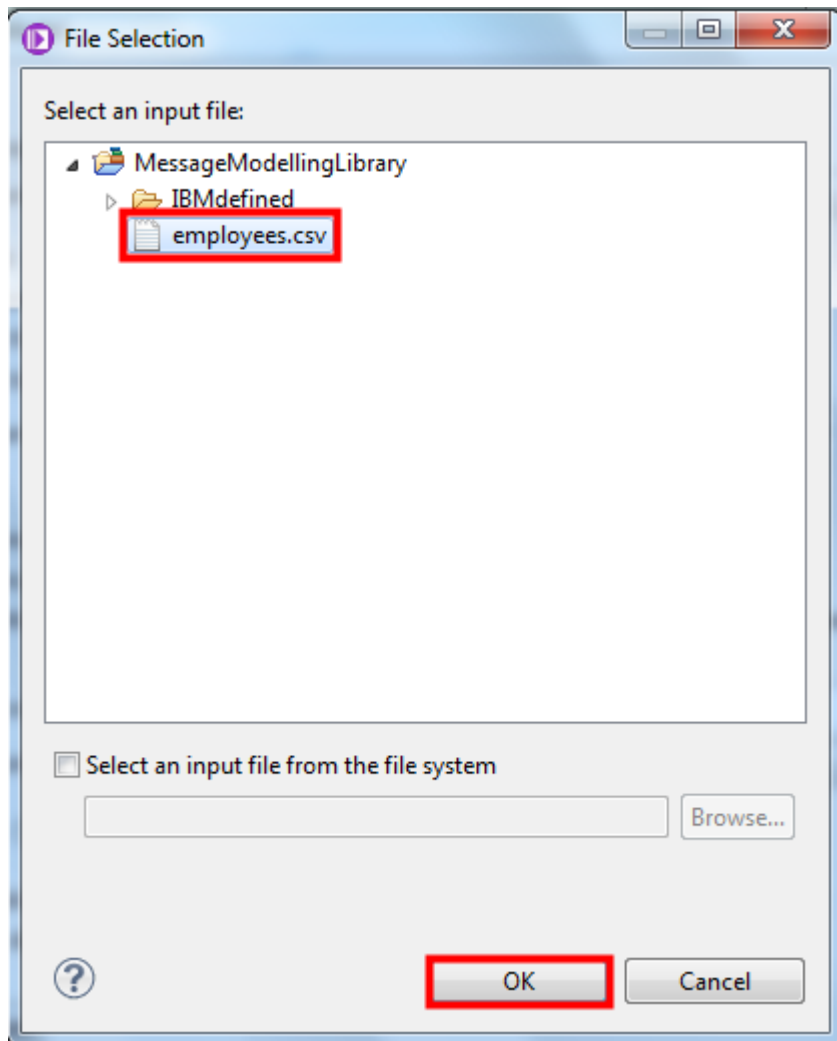
Runtime encoding options
Provide runtime values for properties which have been configured in the model to be dynamically set.
[More...](#)
Encoding (code page): UTF-8
Floating point format: IEEE Non-Extended
Byte order: ☐ Little endian ☒ Big endian

Runtime validation
☐ Validate data against schema [More...](#)

Restore Defaults

OK Cancel

- __4. Select the **employees.csv** file under the **MessageModellingLibrary** library.
- __5. Press the **OK** button.



__6. Press the **OK** button.

Test Parse Model

Message
Select message for testing. [More...](#)
Message name:* Employee

Parser Input
Select content to be parsed against schema.
☐ Content from 'DFDL Test - Serialize' view
☒ Content from a data file
Input file name:* /MessageModellingLibrary/employees.csv [Browse...](#)

Specify runtime configuration.

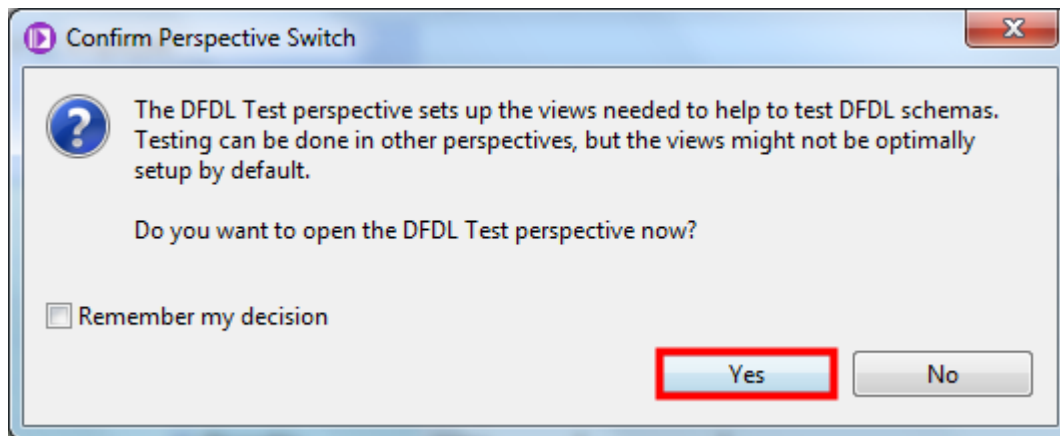
Runtime encoding options
Provide runtime values for properties which have been configured in the model to be dynamically set.
[More...](#)
Encoding (code page): UTF-8
Floating point format: IEEE Non-Extended
Byte order: ☐ Little endian ☒ Big endian

Runtime validation
☐ Validate data against schema [More...](#)

[Restore Defaults](#)

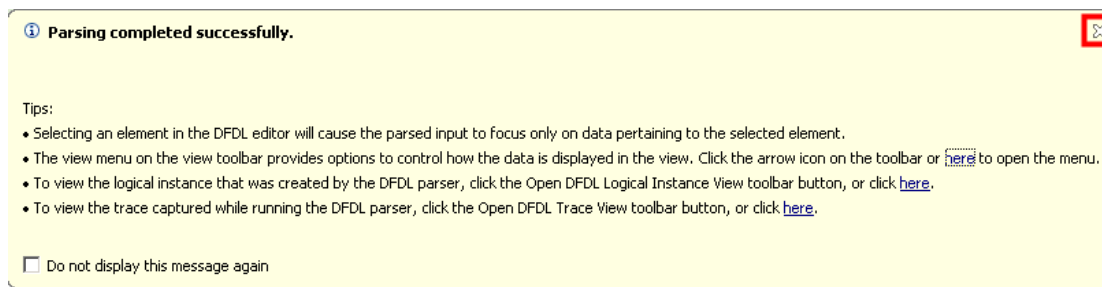
OK Cancel

__7. Press the **Yes** button to continue.



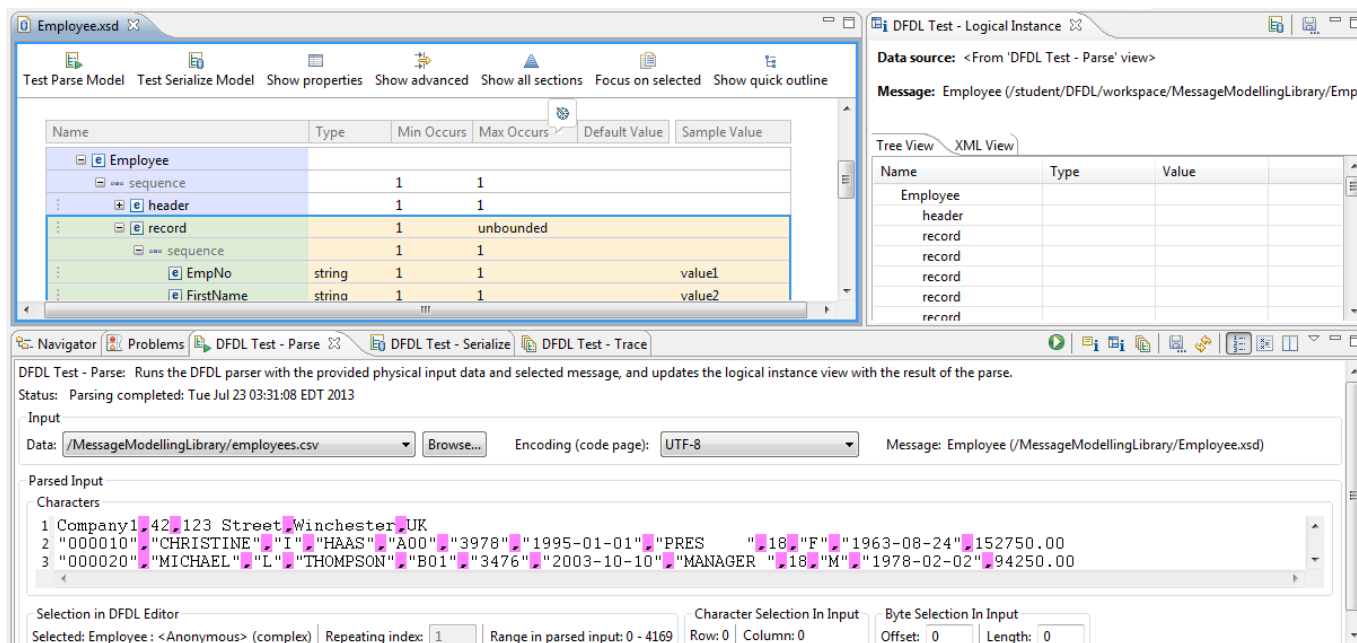
A pop-up dialog should display the results of the parsing test.

__8. Click the **X** to close the dialog box.



The DFDL Test perspective has several views:

- Parse: Allows you to parse a file using the message model in the editor.
- Trace: Has a detailed trace log of the testing activities. Very helpful to find the cause of errors.
- Logical Instance: Gives you a view of the parsed message tree.
- Serialize: Allows you to generate a file from the message model.

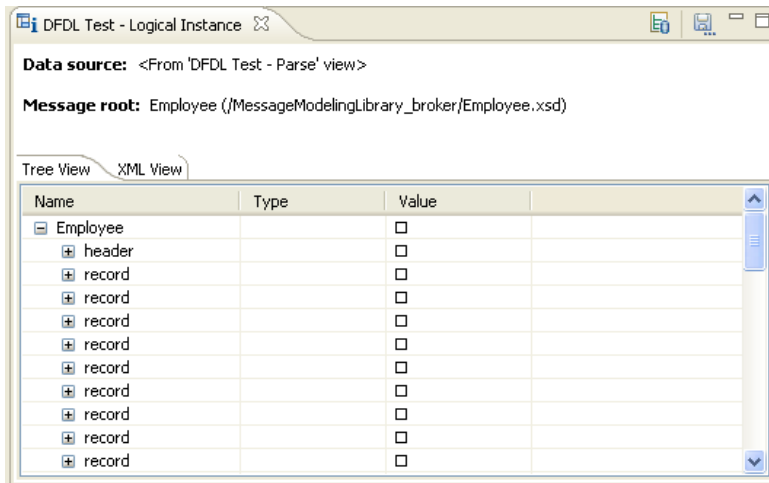


The input text has been highlighted to differentiate the separators (",") that the parser has detected.

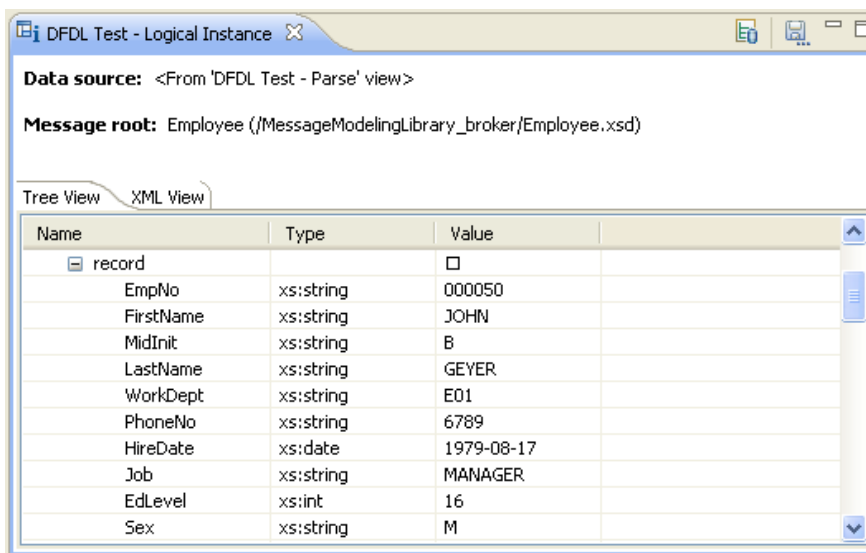
```

Parsed Input
Characters
1 Company1 42 123 Street Winchester UK
2 "000010" "CHRISTINE" "I" "HAAS" "A00" "3978" "1995-01-01" "PRES" "18" "F" "1963-08-24" "152750.00
3 "000020" "MICHAEL" "L" "THOMPSON" "B01" "3476" "2003-10-10" "MANAGER" "18" "M" "1978-02-02" "94250.00
4 "000030" "SALLY" "A" "KWAN" "C01" "4738" "2005-04-05" "MANAGER" "20" "F" "1971-05-11" "98250.00
5 "000050" "JOHN" "B" "GEYER" "E01" "6789" "1979-08-17" "MANAGER" "16" "M" "1955-09-15" "80175.00
6 "000060" "IRVING" "F" "STERN" "D11" "6423" "2003-09-14" "MANAGER" "16" "M" "1975-07-07" "72250.00
  
```

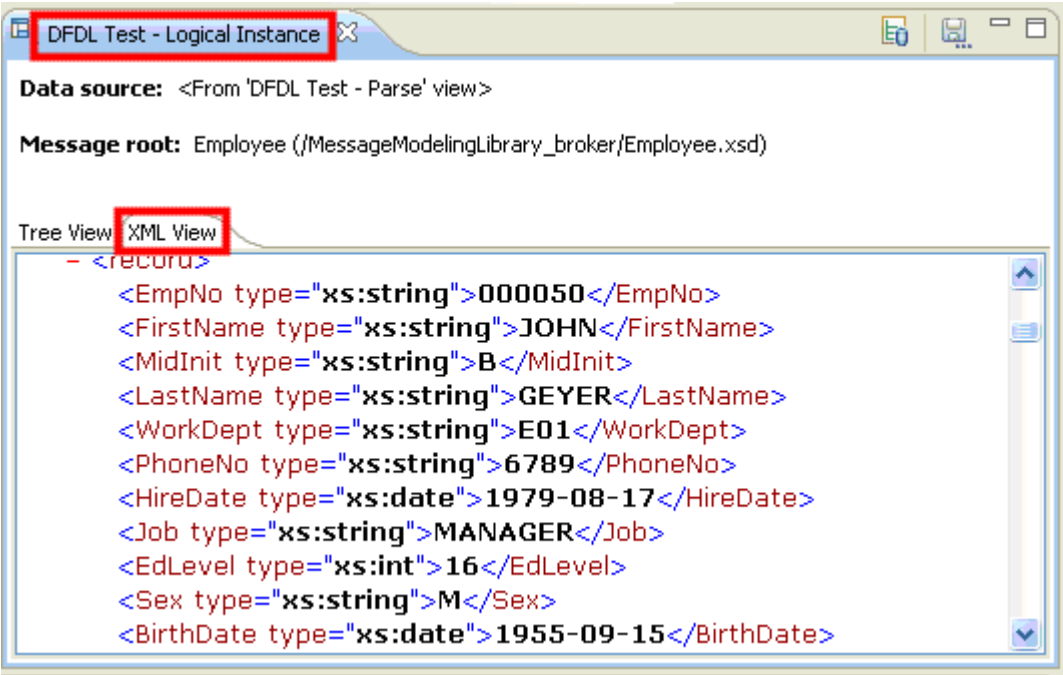
The Logical Instance view is located in the top right corner. This shows the parsed message tree for the employees.csv file



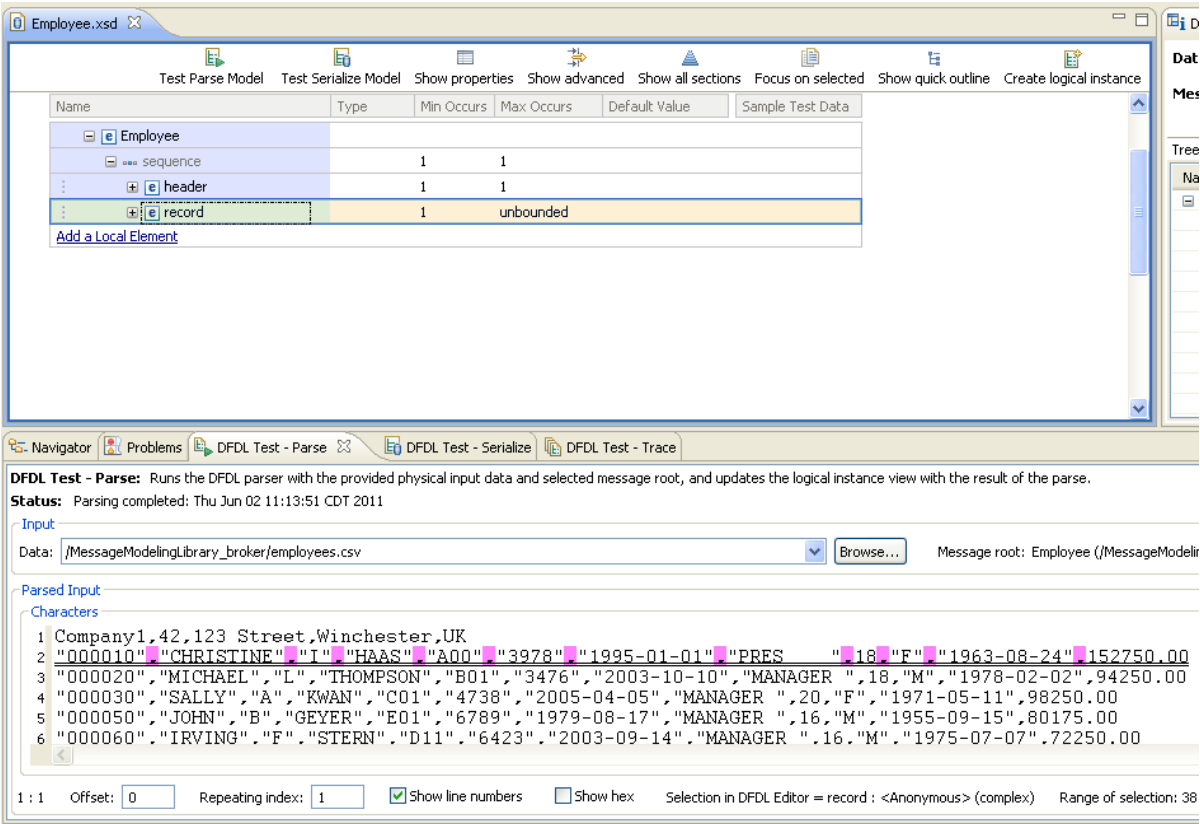
Expand a few records to check that the parsing was correct.



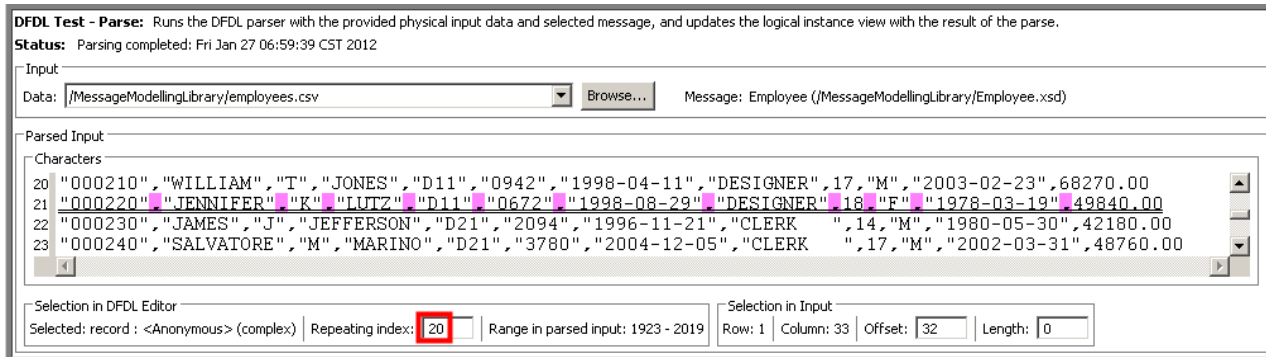
The **XML View** tab shows the same parsed records in XML format.



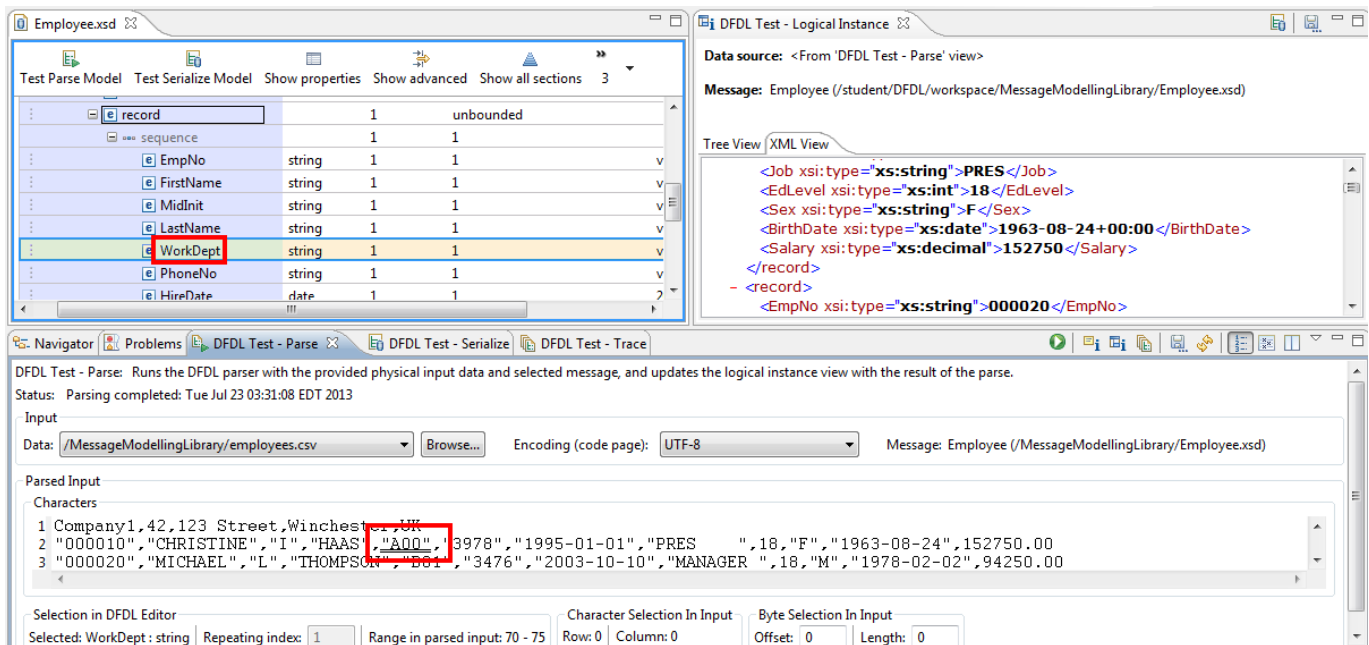
__9. In the **Employee** message model, click on the **Record** element. The first record in the input text will be underlined.



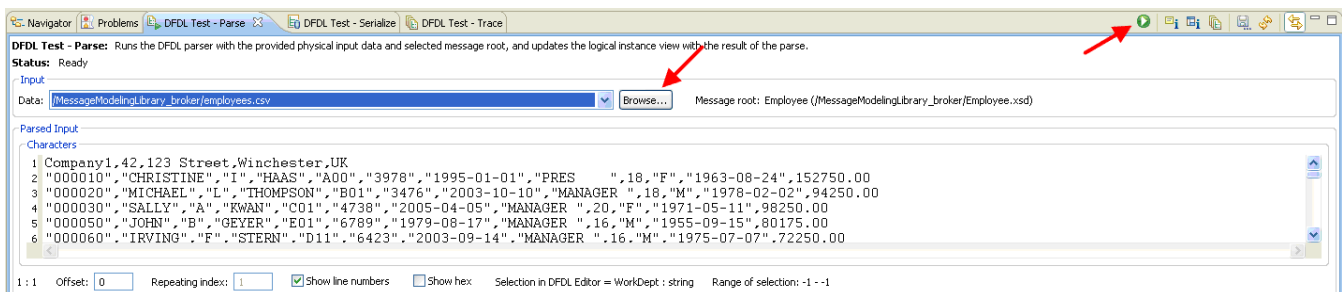
To go to another record in the input text, just change the **Repeating Index** number.



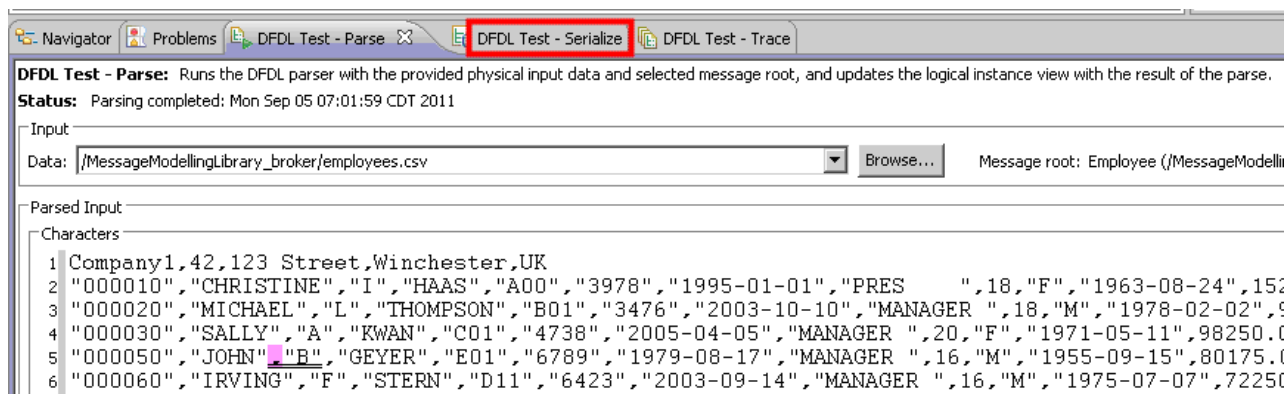
__10. Click on any element in the message model to see where it is located in the input text.



The parser can be run against other input text by clicking the **Browse** button in the input section, selecting the file, and clicking the **Run Parser** button.

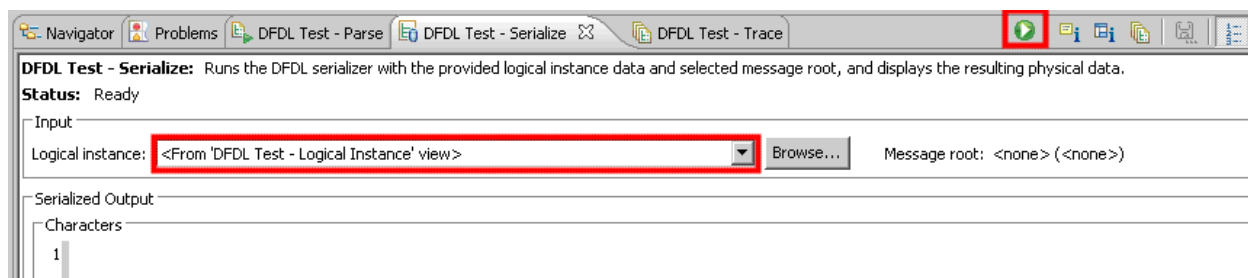


__11. Select the **DFDL Test – Serialize** tab

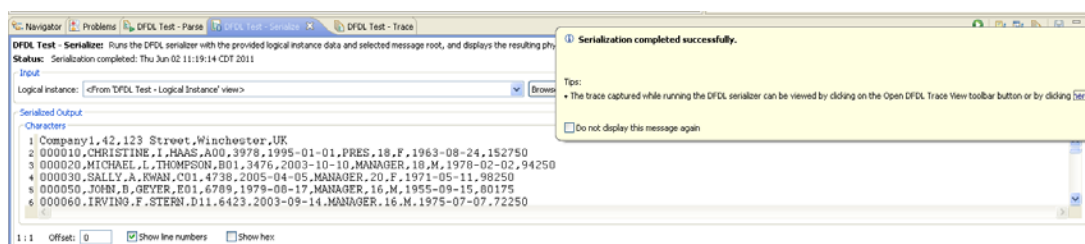


__12. Select **<From "DFDL Test – Logical Instance" view>** for the **Logical Instance**, using the drop down list.

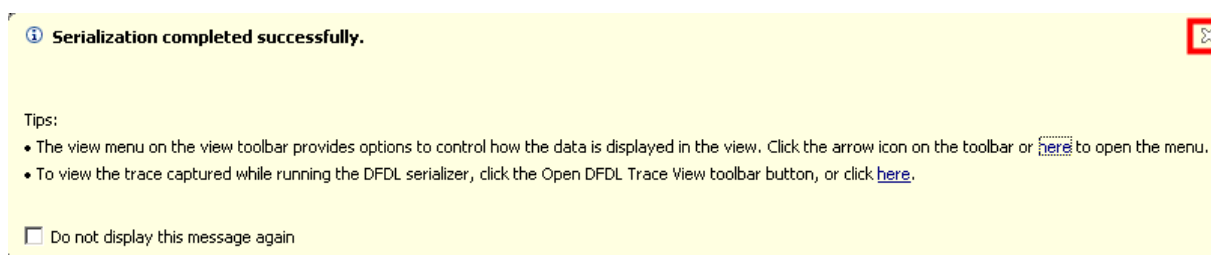
__13. Press the **Run Serializer** icon.



__14. The serializer creates a text file from the previously parsed message tree in memory.

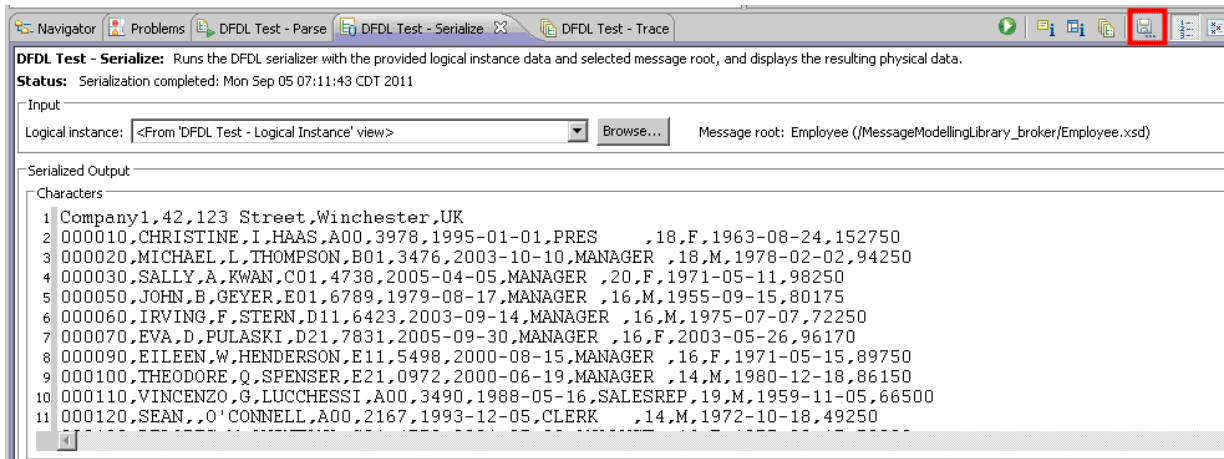


__15. Click the **X** icon to dismiss the results dialog.



The results of the serialization test should appear. The output will now be saved as a file.

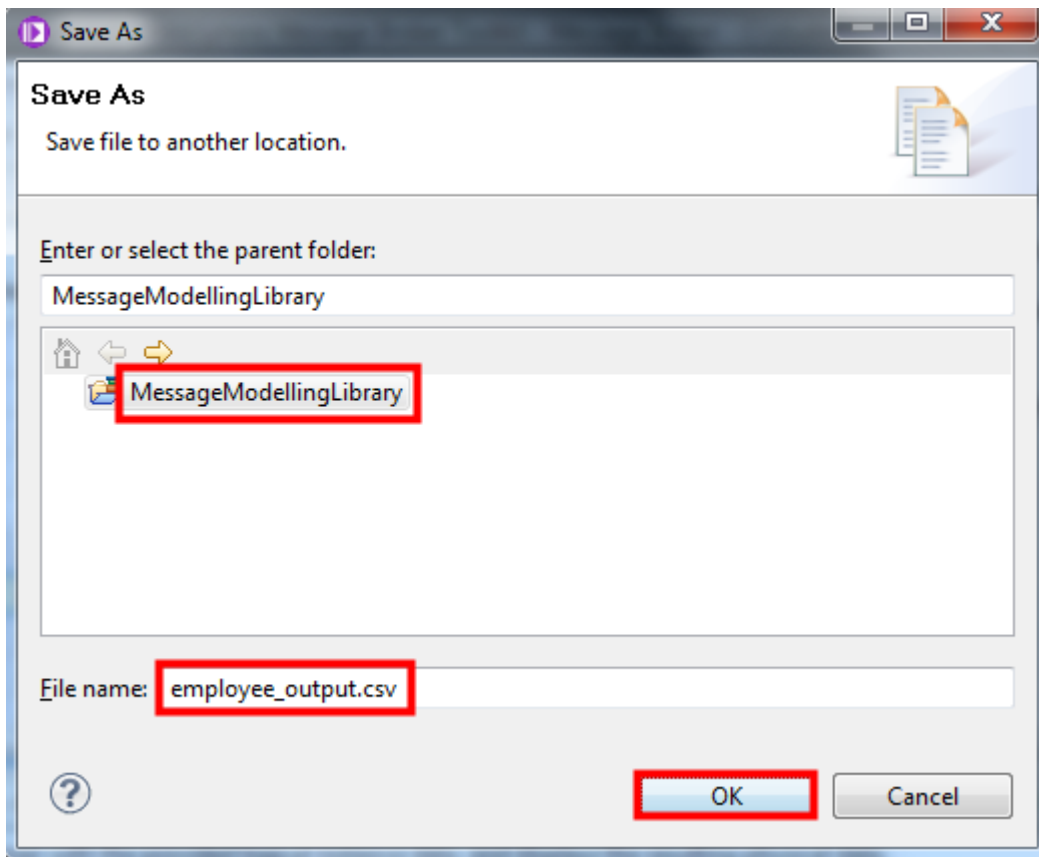
__16. Press the **Save to File** button (diskette icon).



__17. Select the **MessageModellingLibrary** as the parent folder.

__18. Enter **employee_output.csv** as the File name.

__19. Press the **OK** button.

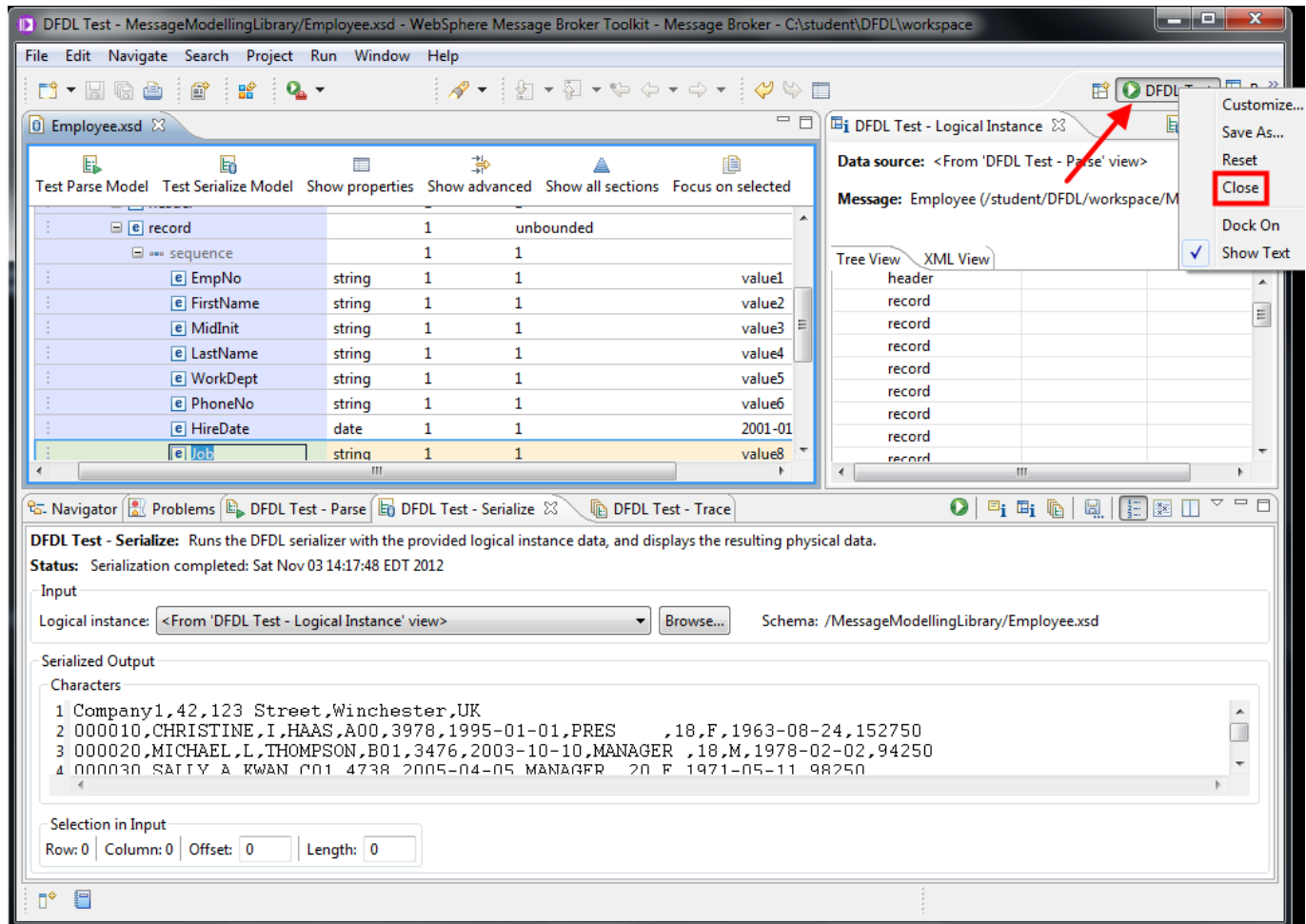


Close the **DFDL Test** perspective.

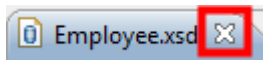
__20. Select the **DFDL Test** perspective tab in the upper right corner.

__21. Press the right mouse button.

__22. Select **Close** from the menu.



__23. Close the **Employee.xsd** tab.



This lab has provided a brief introduction to some of the new capabilities that have been provided with the DFDL parser. The DFDL domain should be the parser of choice for all non-XML data formats.

This is the end of Lab 2.

Lab 3 Graphical Data Mapper

3.1 Introduction to mapping

IBM Integration Bus Version 9 includes the high performing IBM Graphical Data Mapper (GDM), a newly upgraded mapping editor. It allows you to visually map and transform source to target data, and is based on common mapping tooling across all WebSphere integration products. GDM is also included in other IBM offerings such as InfoSphere® Master Data Management v10, Rational® Application Developer for WebSphere Software v8.5 and Rational Software Architect v8.5. It generates a standards-based format of the mapping function known as Map Specification Language.

GDM allows you to apply transformations to single and multiple elements, with support for conditionals (if then else), loops (for each), functions and more. It supports database mapping sources and targets for routing and enrichment scenarios, including support for database selects, inserts, updates, deletes and the ability to directly access stored procedures. It complements and supports existing transformation languages, including the ability to call user defined transformations in Java, ESQL and XPATH.

In IBM Integration Bus V9, the GDM supports migration of pre-v8 maps. Most sophisticated maps can be converted in a single step. The editor provides enhanced feedback about conversion to assist user understanding.

3.2 Preparation – Build the SAMPLE connection

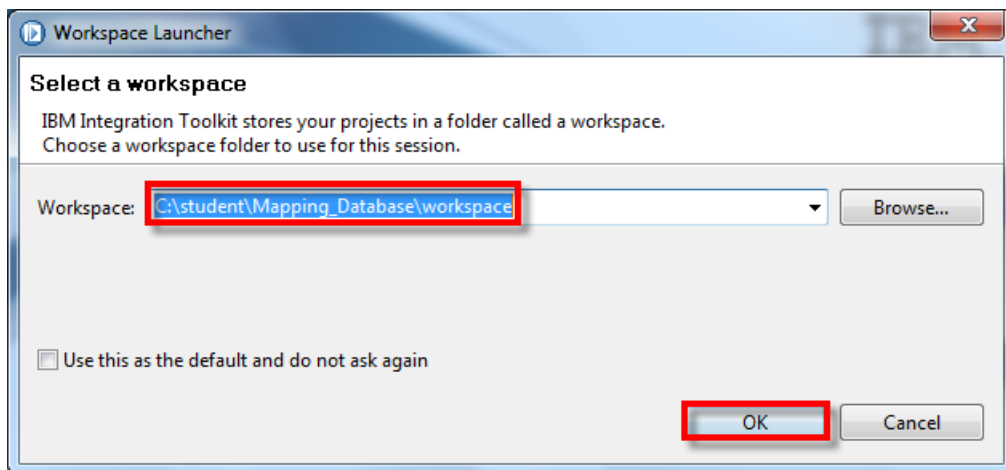
NOTE: This is the first step required no matter which part of the Mapping lab you plan to build. It is used by all further sections of the Mapping lab.

The Mapping Node uses JDBC to connect to relational databases. The appendix of the lab describes the JDBC configuration that is necessary for the Integration Bus Runtime to access the SAMPLE database from a Mapping node.

The first task is to create a Library that will be used as a container for the required database connections and any Message Models used by the labs.

This lab will start from a blank workspace.

- __1. Select **File→Switch Workspace→Other**.
- __2. Enter **C:\student\Mapping_Database\workspace** for the Workspace.



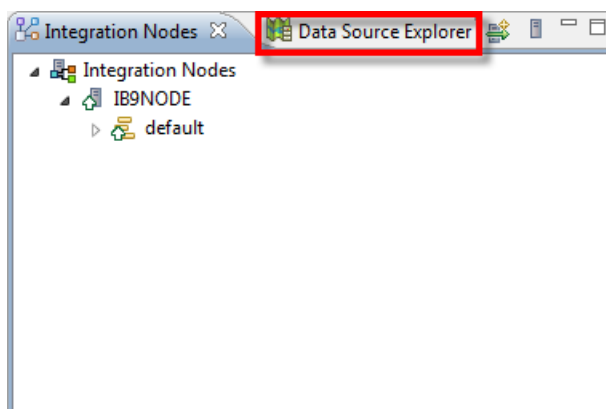
- __3. Press the **OK** button to continue.
- __4. Click on the **Go to the Integration Toolkit** link to close the **Welcome** screen.



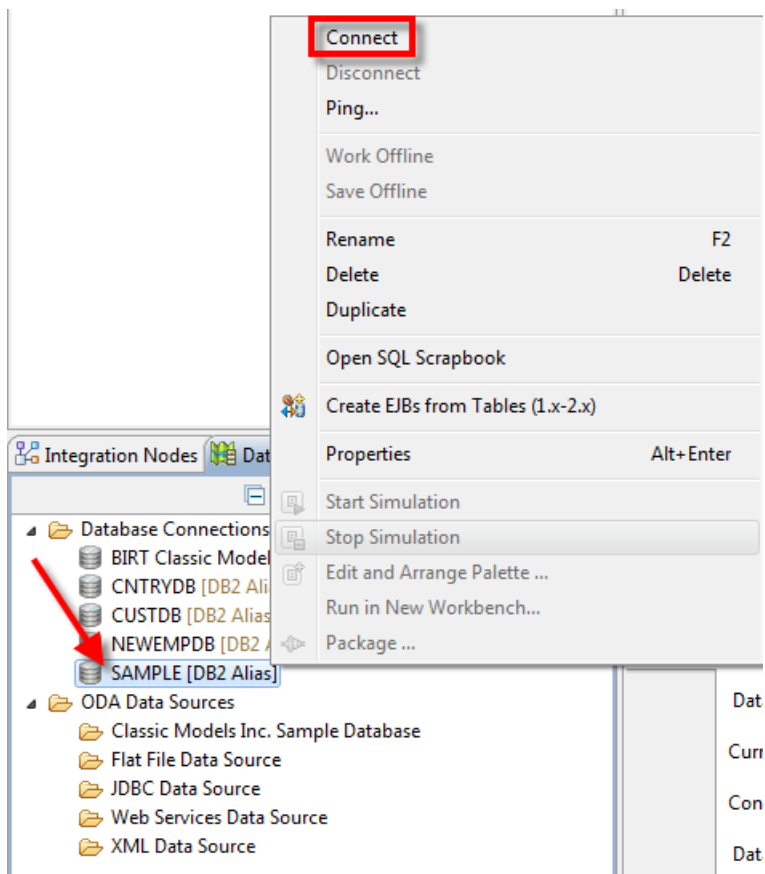
The lab will retrieve employee information from the EMPLOYEE table in the SAMPLE DB2® database. We therefore need to create a Database Definition project so that the Message Flow can access the database.

The database connection for SAMPLE already exists. So we will connect to it so we can use it.

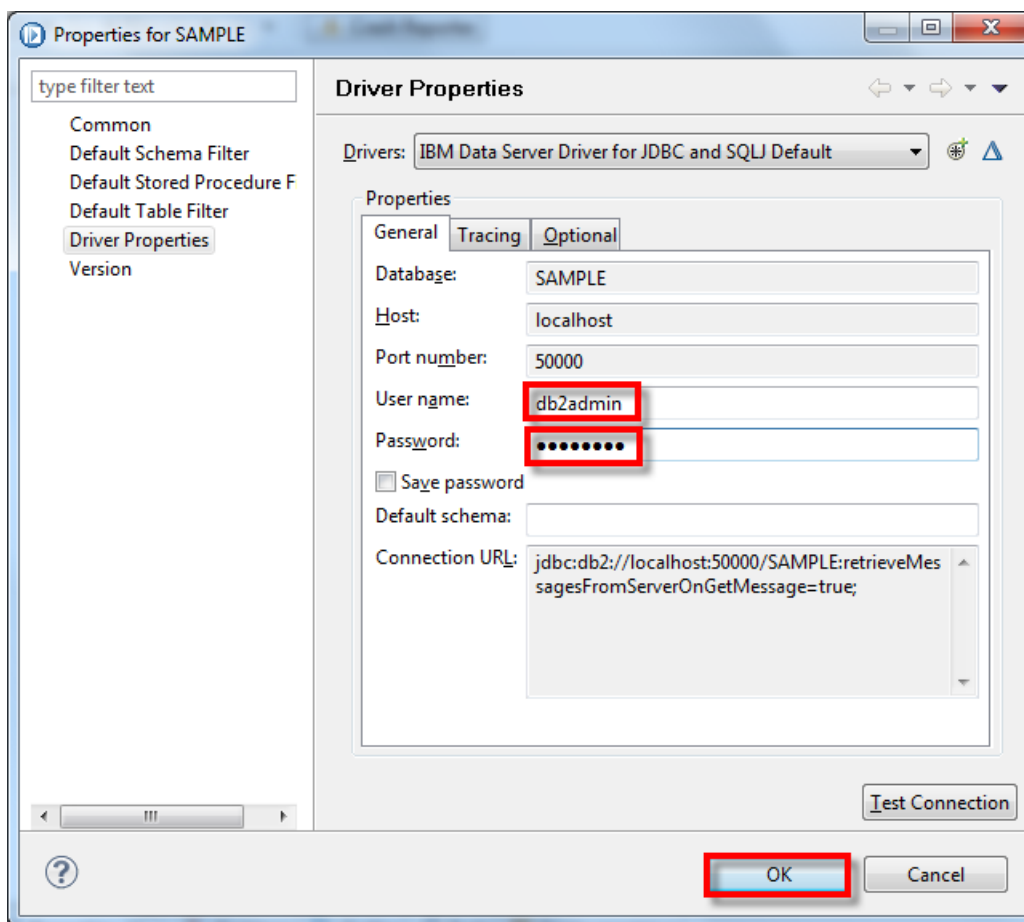
- __5. Click on the **Data Source Explorer** view to open it.



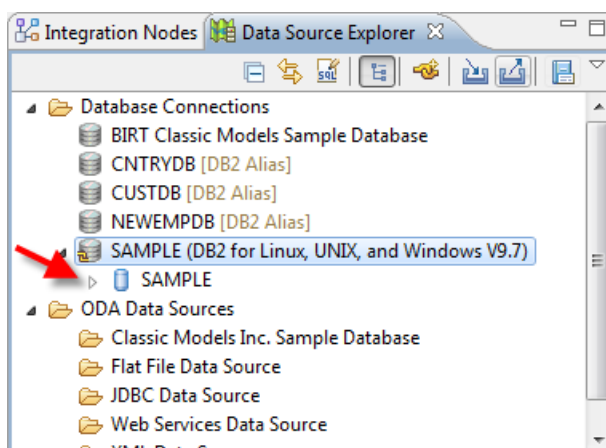
- ___6. Right-click the **SAMPLE** Database Connection, and select **Connect** from the menu.



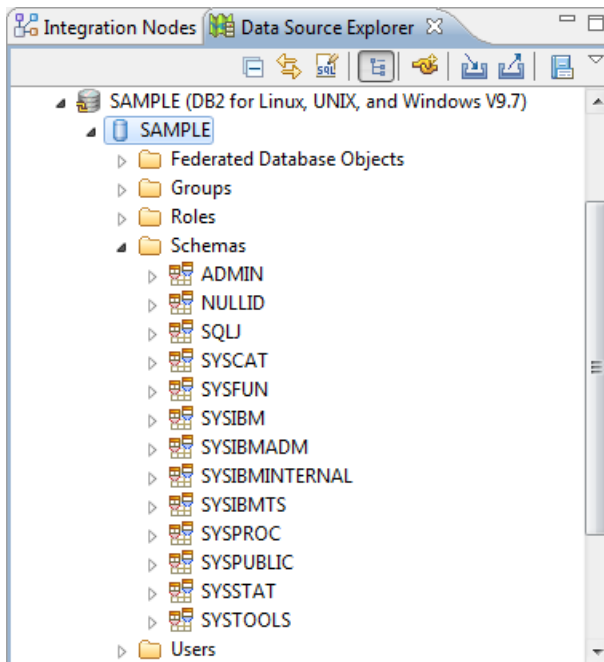
- ___7. Set the User name to **db2admin** and Password to **db8admin**. Click **OK**.



- ___8. A JDBC connection will be established to the SAMPLE database, and you will see the **SAMPLE** database with a blue “connected” database icon.

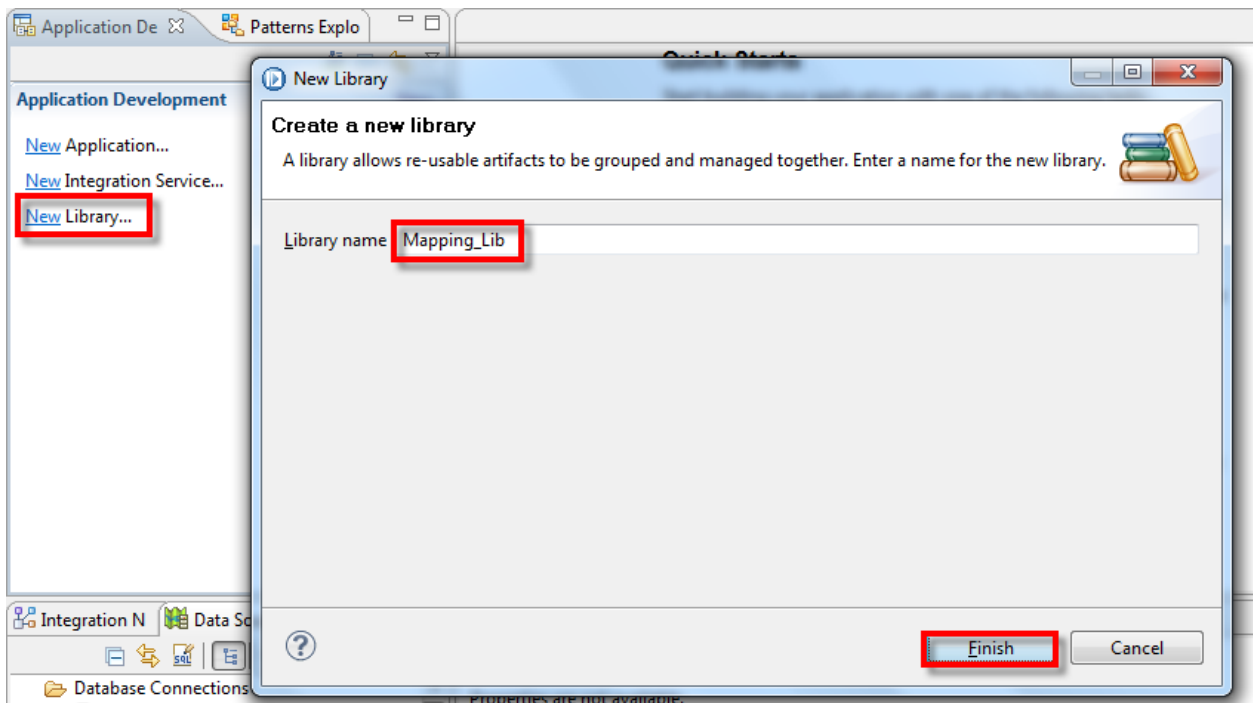


- ___9. If you open the twistie next to **SAMPLE**, you can explore the different categorized folders of database assets and users that you can further explore, with the ability to get all the way down to the definitions and contents. All that is needed for now is to establish this connection.

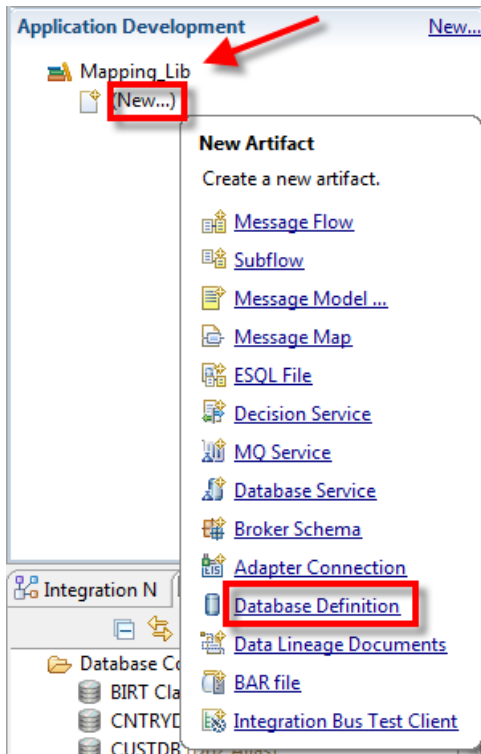


We can now build our database definition file.

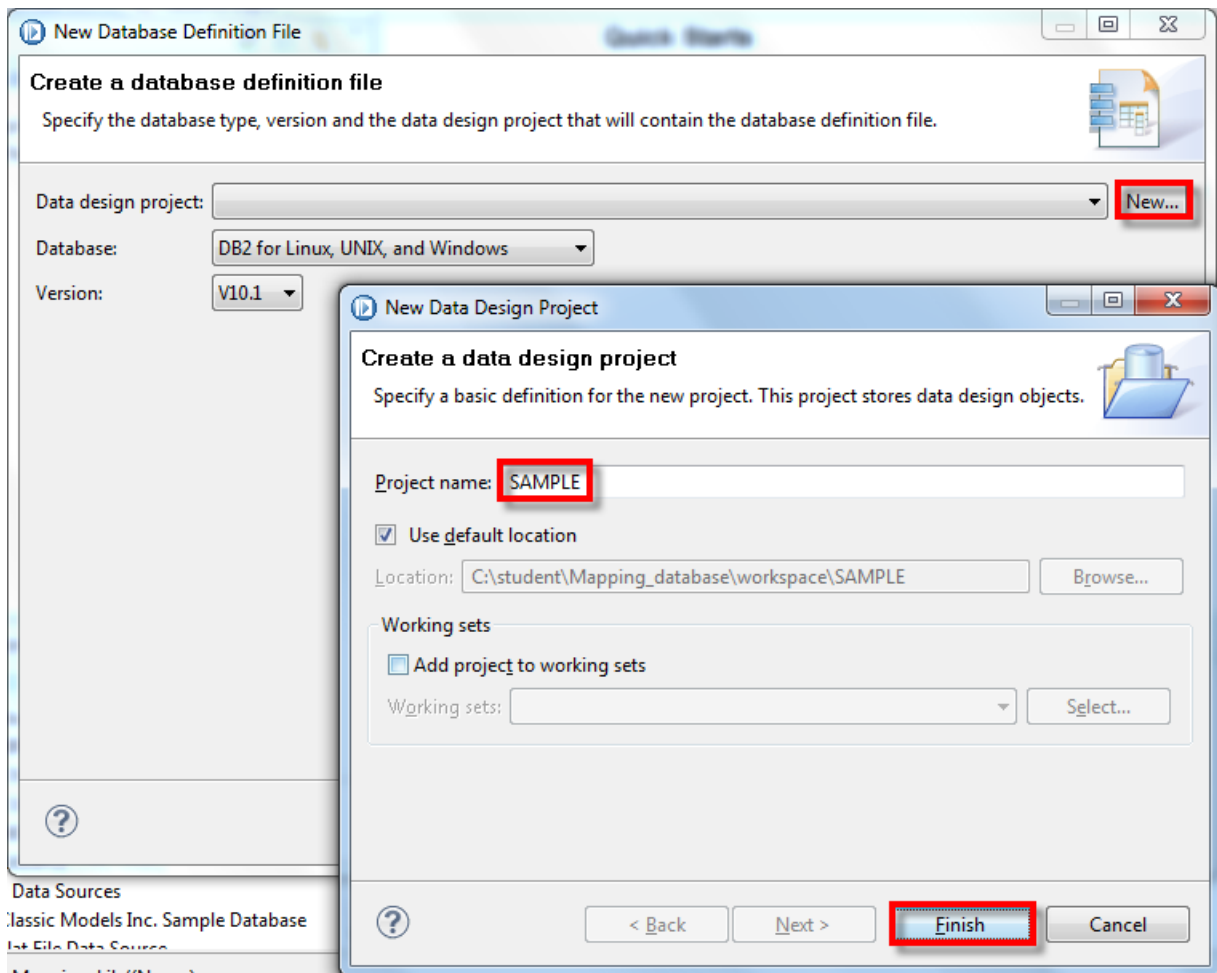
- ___10. Start by creating a new library by clicking the **New Library...** link. Name the new library **Mapping_Lib** and click **Finish**.



___11. In the **Mapping_Lib** library, click **New**, and select **Database Definition**.

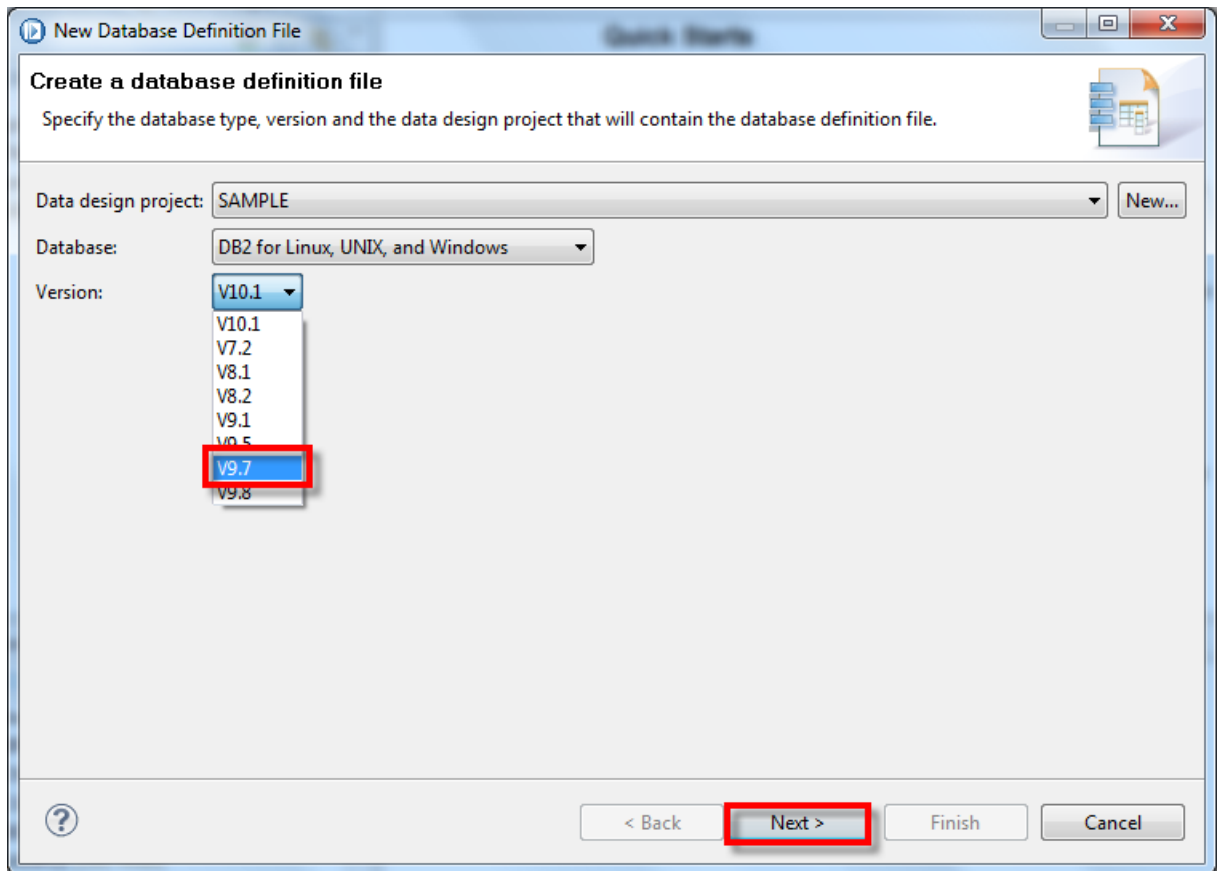


- ___12. Since we do not have an existing project to store this, click **New** to create a new database design project. Name this **SAMPLE**.



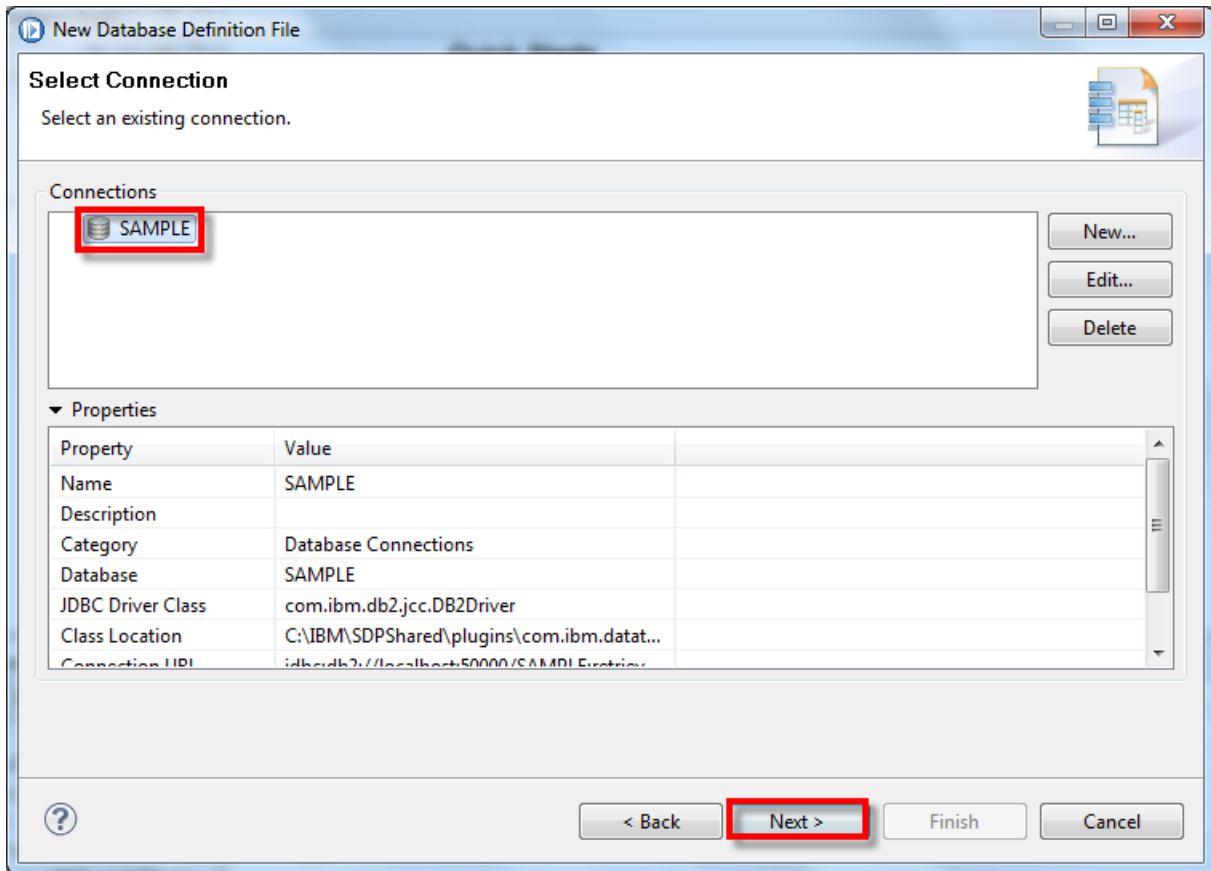
- ___13. Click **Finish**.

- ___14. The Database Design project field has now been populated with the SAMPLE project. Select **V9.7** from the **Version** drop-down box.

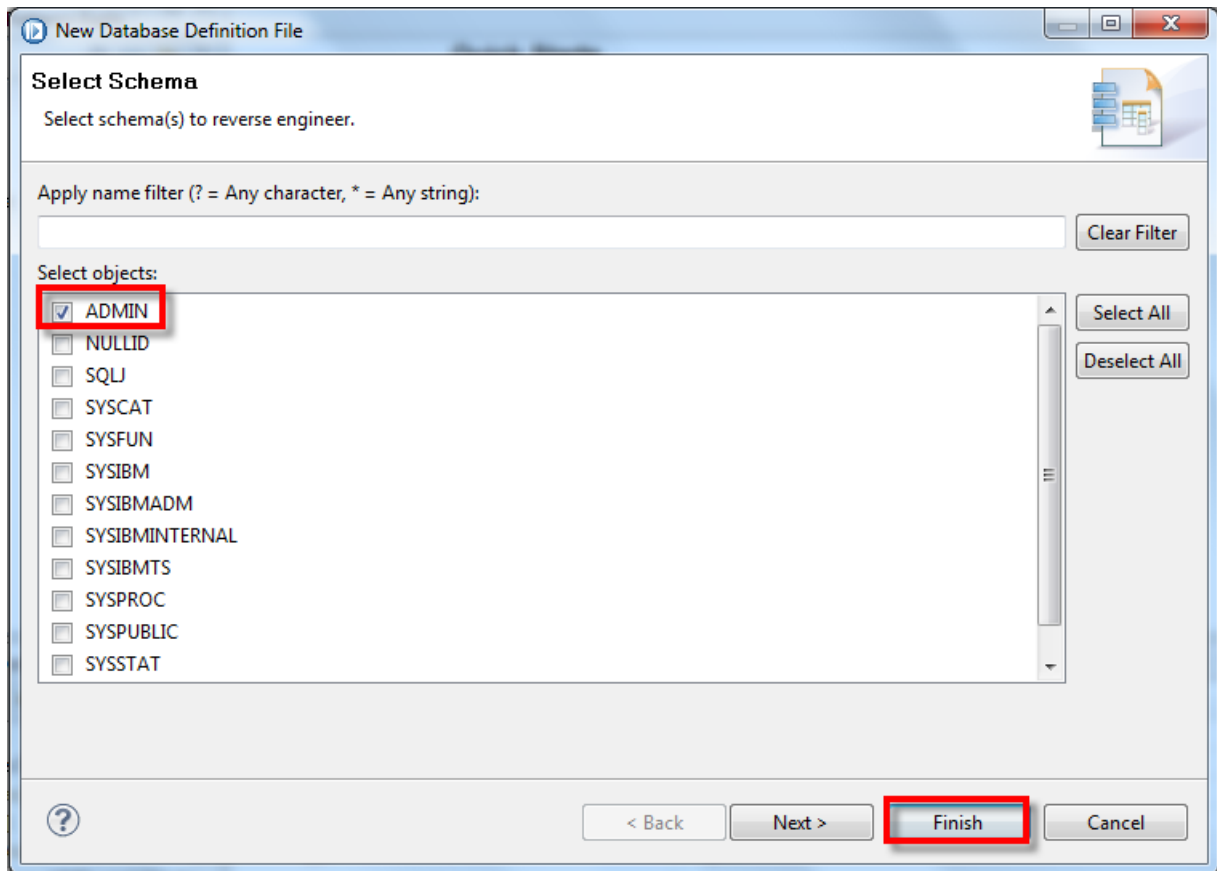


- ___15. Click **Next**.

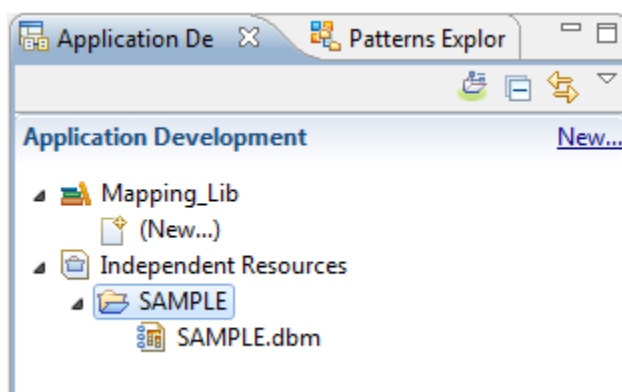
___16. Highlight the **SAMPLE** connection that we have connected to and click **Next**.



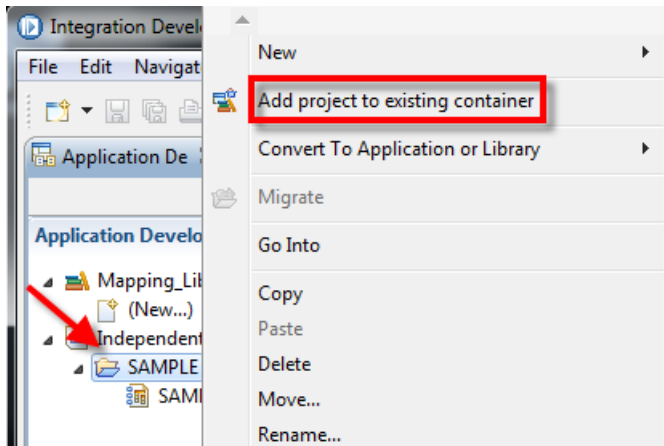
- ___17. Select the **ADMIN** schema (the SAMPLE database was created under the ADMIN schema). Click **Finish**.



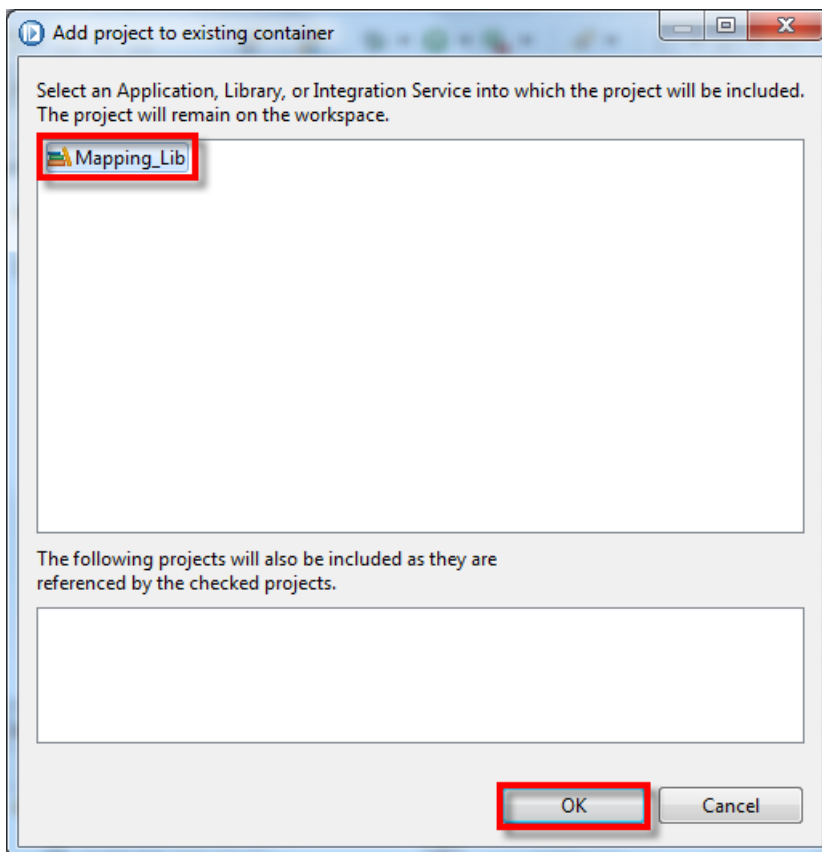
- ___18. The Database Design project has now been created. It has been created as a separate project under the **Independent Resources** in the navigator.



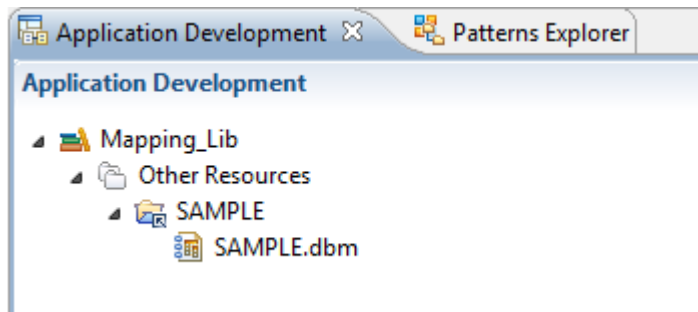
- ___19. Right-click the **SAMPLE** project and select “Add project to existing container”.



- ___20. Select the **Mapping_Lib** library and click **OK**.



- ___21. The Database Definition file, **SAMPLE.dbm**, will now be seen in the **Other Resources** folder in the **Mapping_Lib** library.



- ___22. The database definitions are now available to the application and do not need to be changed further. If it is open, close the **SAMPLE.dbm** editor.

3.3 Using a select in a map

This lab will use the standard DB2 SAMPLE database (EMPLOYEE table) and a Message Model based on a COBOL copybook.

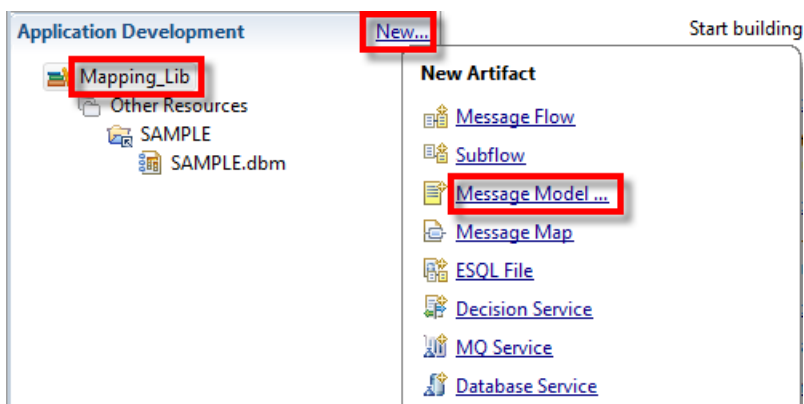
An incoming message in COBOL format will be used to retrieve employee information from the EMPLOYEE table. This information will be used to populate several fields in the COBOL message, before being output from the message flow.

The lab will show you how to build the Database Definition, Message Model, Message Flow and the Mapping Node that will be used to retrieve the database row and build the outgoing message.

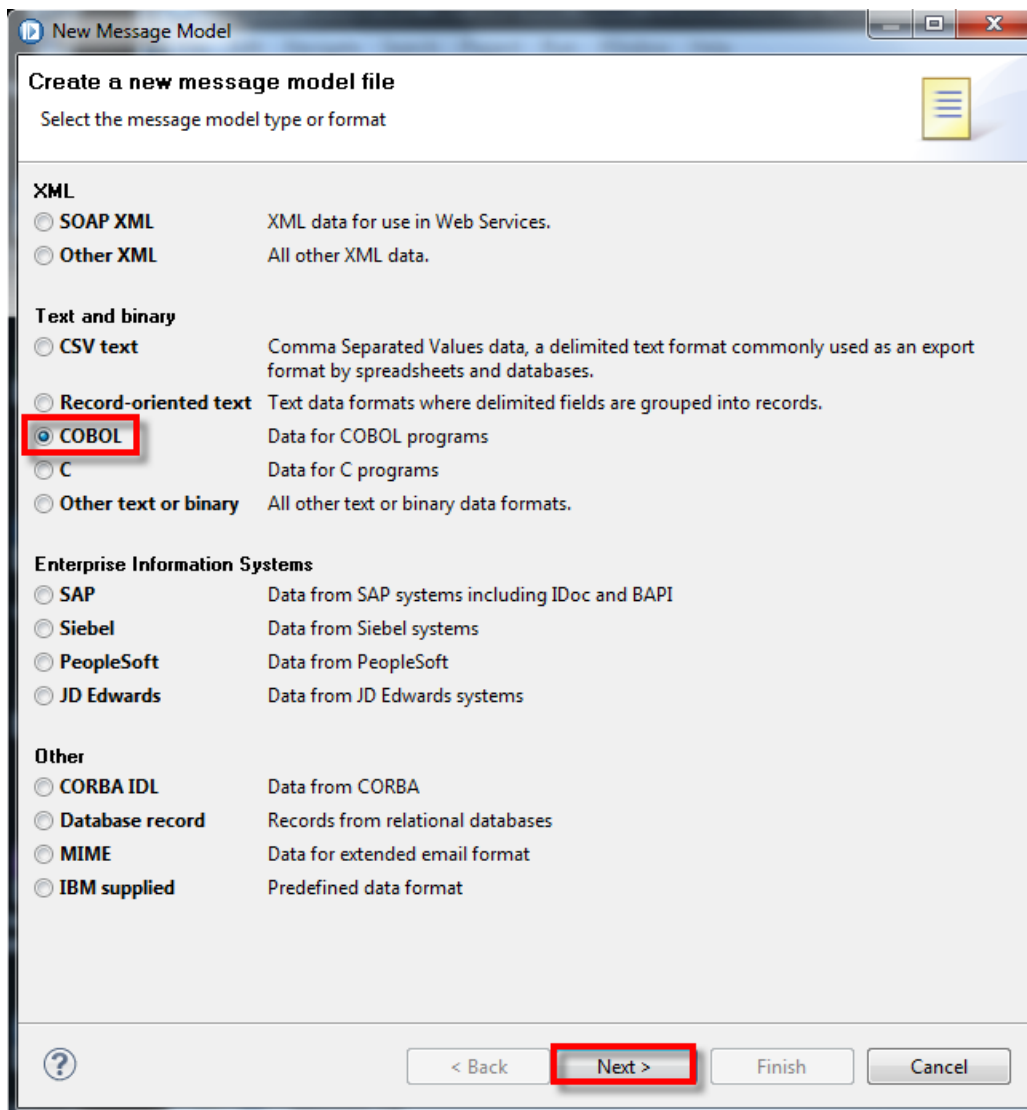
We will now create a new Message Model using the DFDL importer, based on an existing COBOL copybook. The message model will be defined in the Mapping_Lib library.

Be sure you have completed the steps in Section 3.2 to build the SAMPLE database definition.

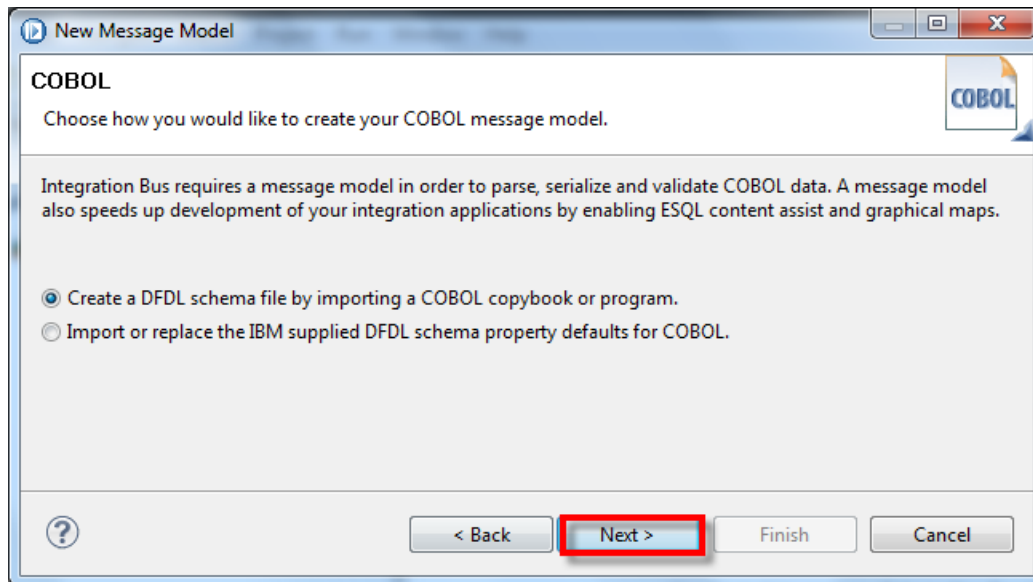
- ___1. Highlight the **Mapping_Lib** library name, and then click **New** and select **Message Model...**



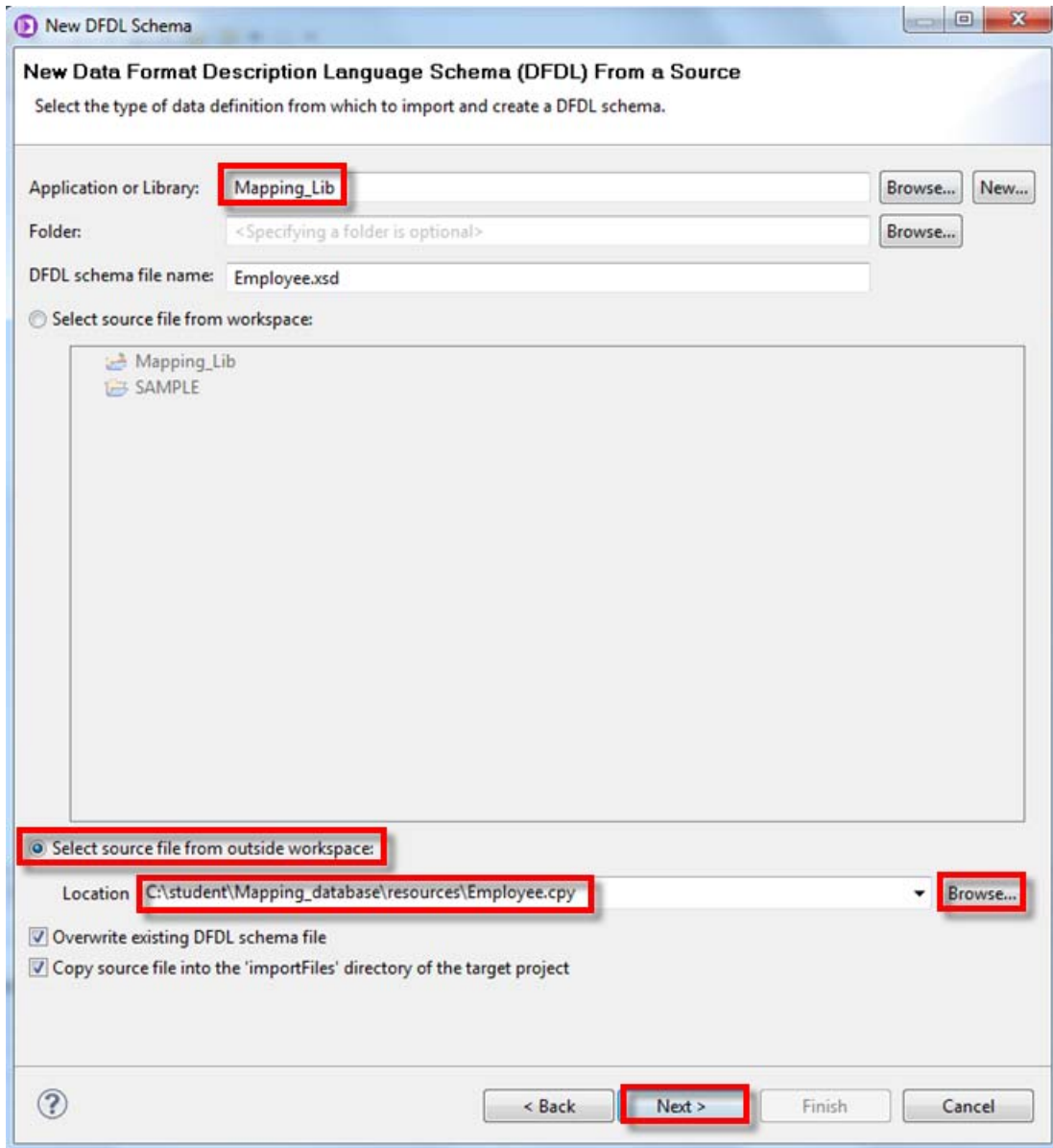
__2. Click the **COBOL** radio button, and click **Next**.



__3. Accept the default selection (Create a DFDL schema file by importing a COBOL copybook program) and click **Next**.



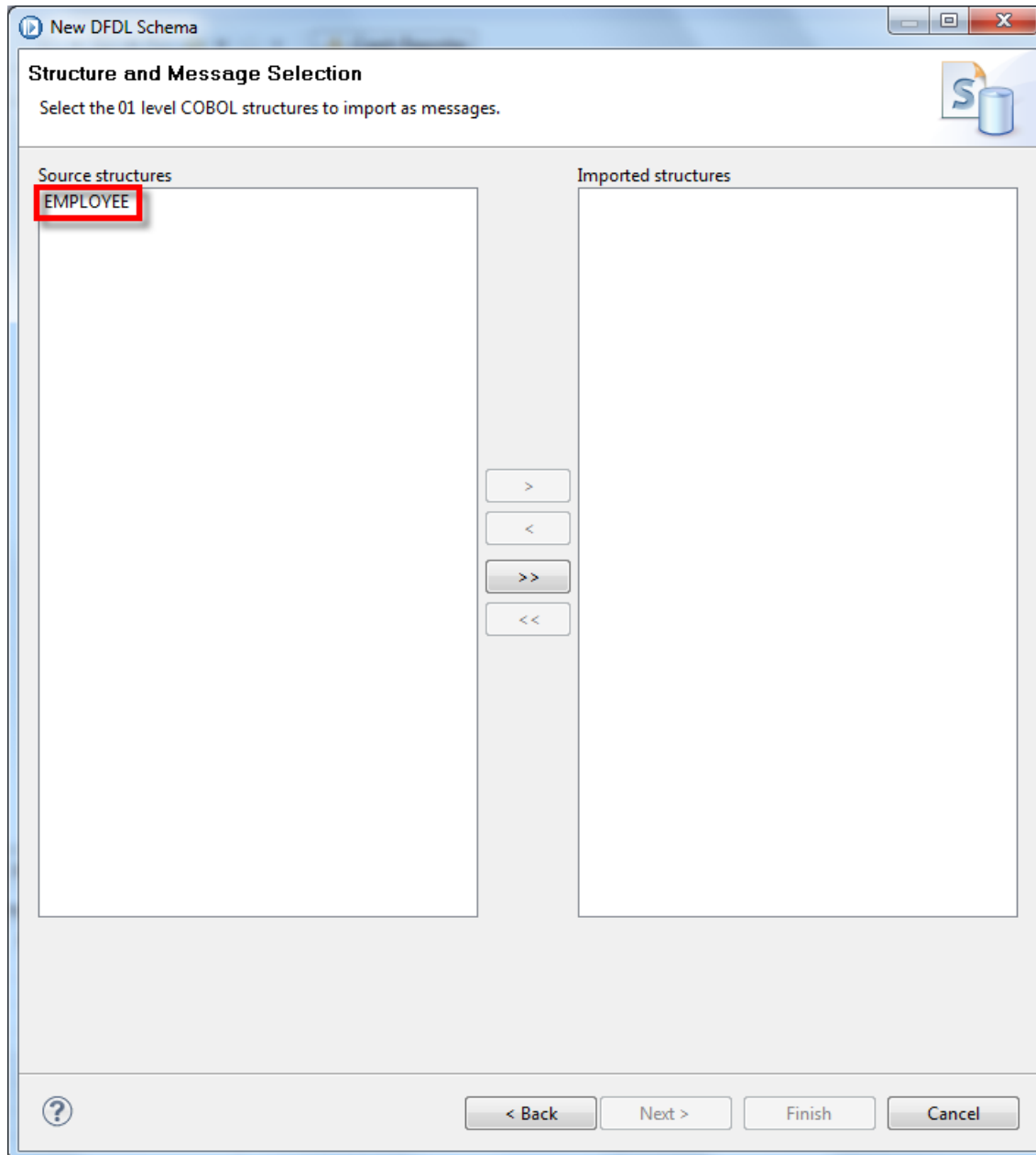
__4. Ensure the Application Name is set to **Mapping_Lib**.



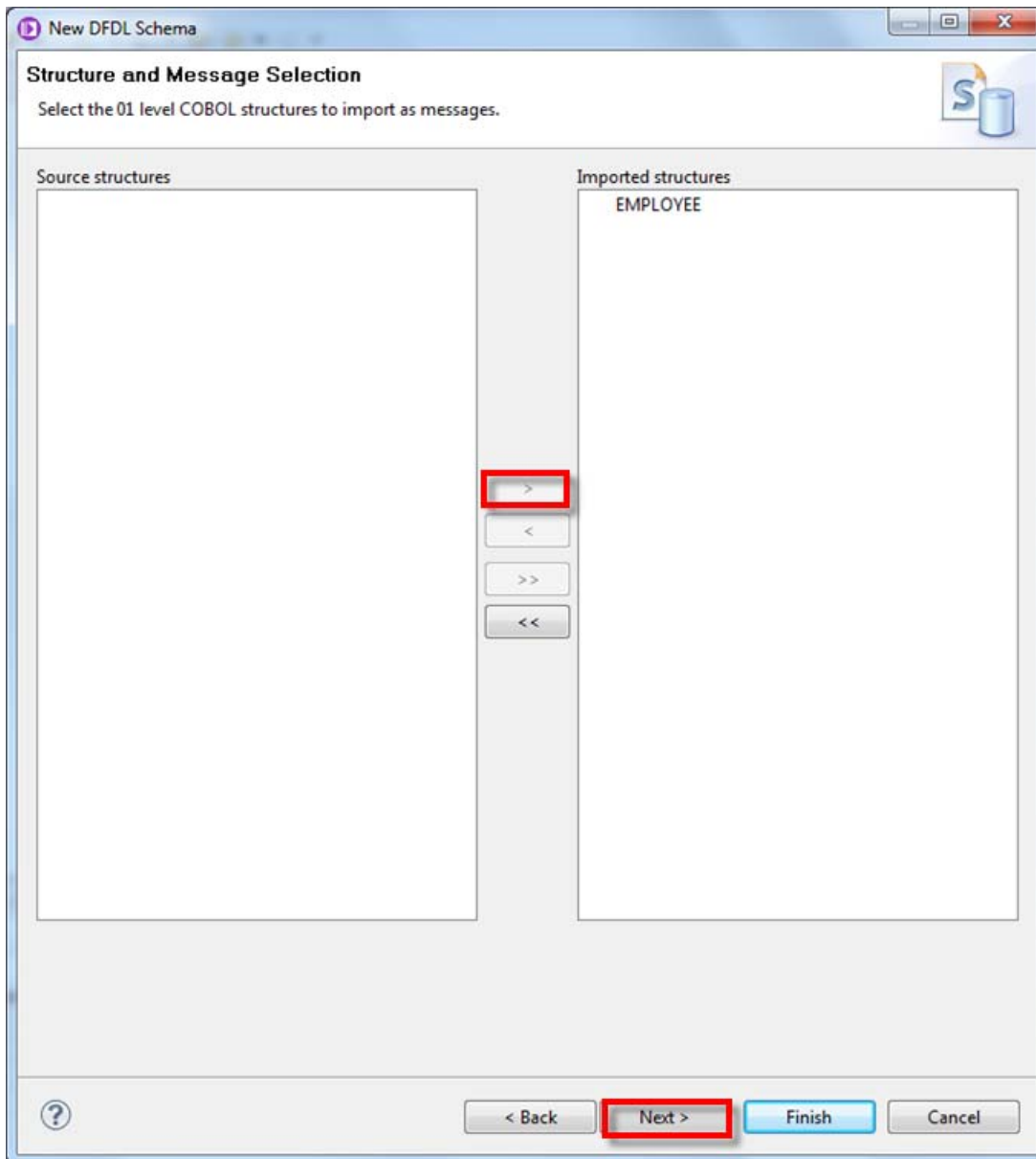
__5. Click “**Select source from outside workspace**”, and set the location of the file to **c:\student\mapping_database\resources\Employee.cpy** (you can use the **Browse** button).

__6. Click **Next**.

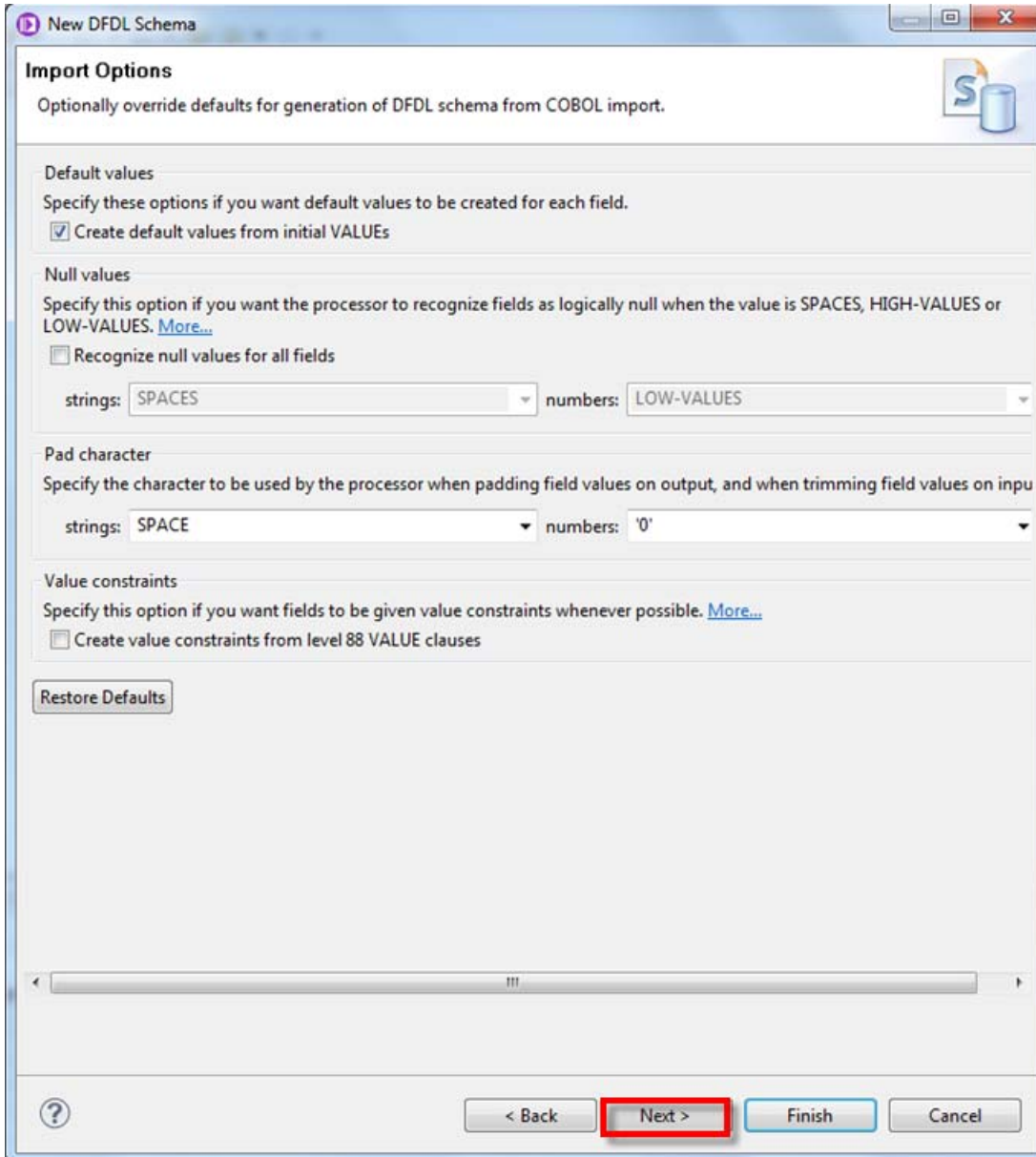
___7. In the Source Structures panel, click on **EMPLOYEE**.



__8. You can then click the right arrow in the middle to move it to the Imported Structures pane. Click **Next**.



__9. Accept all the default values and click **Next**.

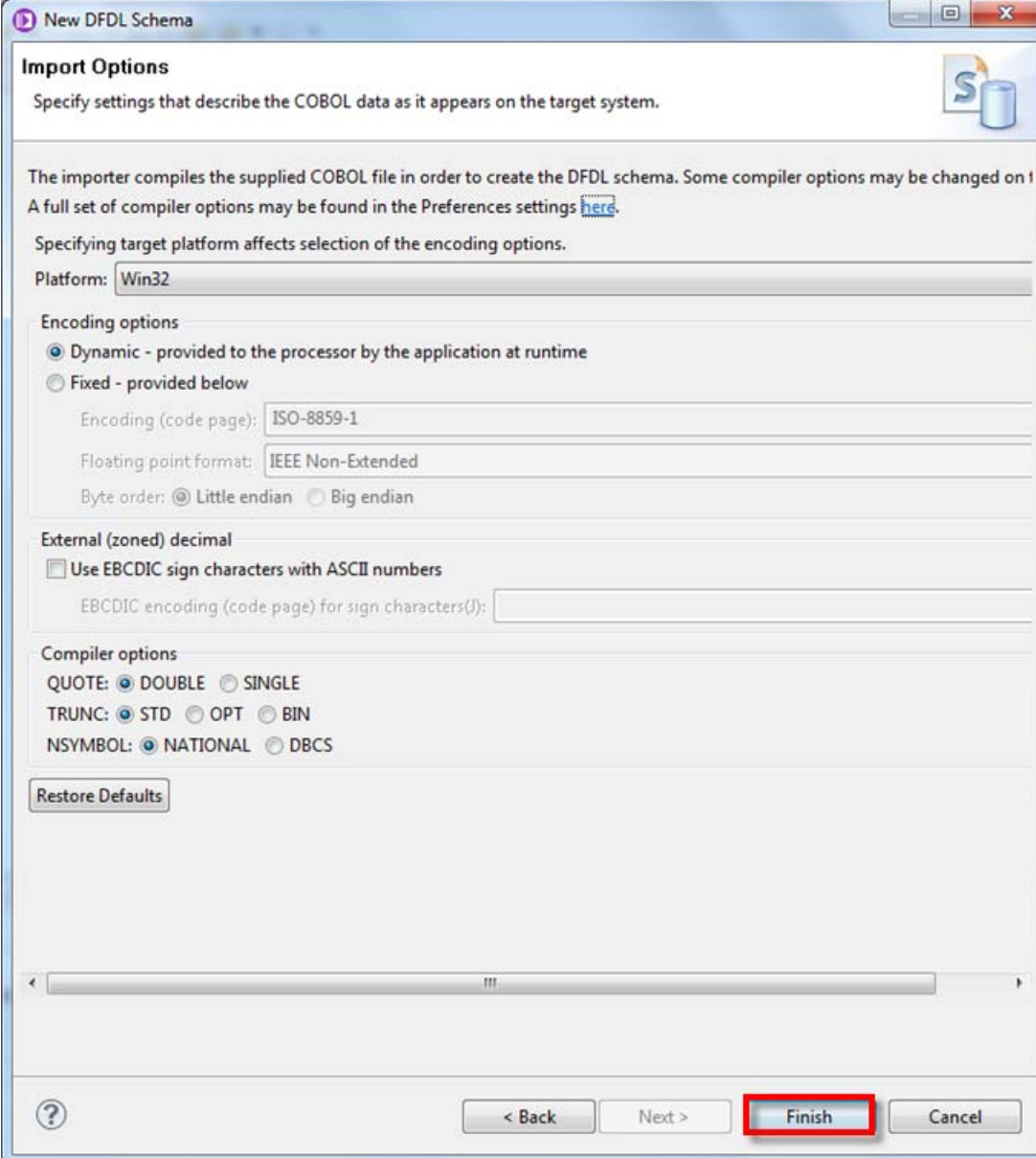


The image shows a Windows-style dialog box titled "New DFDL Schema". The main heading is "Import Options" with a subtitle "Optionally override defaults for generation of DFDL schema from COBOL import." and a small icon of a document with a cylinder. The dialog is divided into several sections:

- Default values:** "Specify these options if you want default values to be created for each field." There is a checked checkbox labeled "Create default values from initial VALUES".
- Null values:** "Specify this option if you want the processor to recognize fields as logically null when the value is SPACES, HIGH-VALUES or LOW-VALUES. [More...](#)" There is an unchecked checkbox labeled "Recognize null values for all fields". Below this are two dropdown menus: "strings:" set to "SPACES" and "numbers:" set to "LOW-VALUES".
- Pad character:** "Specify the character to be used by the processor when padding field values on output, and when trimming field values on input." There are two dropdown menus: "strings:" set to "SPACE" and "numbers:" set to "'0'".
- Value constraints:** "Specify this option if you want fields to be given value constraints whenever possible. [More...](#)" There is an unchecked checkbox labeled "Create value constraints from level 88 VALUE clauses".

At the bottom left is a "Restore Defaults" button. At the bottom right are four buttons: "< Back", "Next >" (which is highlighted with a red rectangle), "Finish", and "Cancel". A scrollbar is visible in the middle of the dialog.

___10. Accept all the default values and click **Finish**.



The image shows a Windows-style dialog box titled "New DFDL Schema". It contains several sections for configuring COBOL data import settings. The "Platform" is set to "Win32". Under "Encoding options", "Dynamic" is selected. "Encoding (code page)" is "ISO-8859-1", "Floating point format" is "IEEE Non-Extended", and "Byte order" is "Little endian". Under "External (zoned) decimal", the checkbox "Use EBCDIC sign characters with ASCII numbers" is unchecked. Under "Compiler options", "QUOTE" is "DOUBLE", "TRUNC" is "STD", and "NSYMBOL" is "NATIONAL". A "Restore Defaults" button is present. At the bottom, there are four buttons: "< Back", "Next >", "Finish" (highlighted with a red rectangle), and "Cancel".

New DFDL Schema

Import Options
Specify settings that describe the COBOL data as it appears on the target system.

The importer compiles the supplied COBOL file in order to create the DFDL schema. Some compiler options may be changed on the target system. A full set of compiler options may be found in the Preferences settings [here](#).

Specifying target platform affects selection of the encoding options.

Platform: Win32

Encoding options

- ☒ Dynamic - provided to the processor by the application at runtime
- ☐ Fixed - provided below

Encoding (code page): ISO-8859-1

Floating point format: IEEE Non-Extended

Byte order: ☒ Little endian ☐ Big endian

External (zoned) decimal

☐ Use EBCDIC sign characters with ASCII numbers

EBCDIC encoding (code page) for sign characters(J):

Compiler options

QUOTE: ☒ DOUBLE ☐ SINGLE

TRUNC: ☒ STD ☐ OPT ☐ BIN

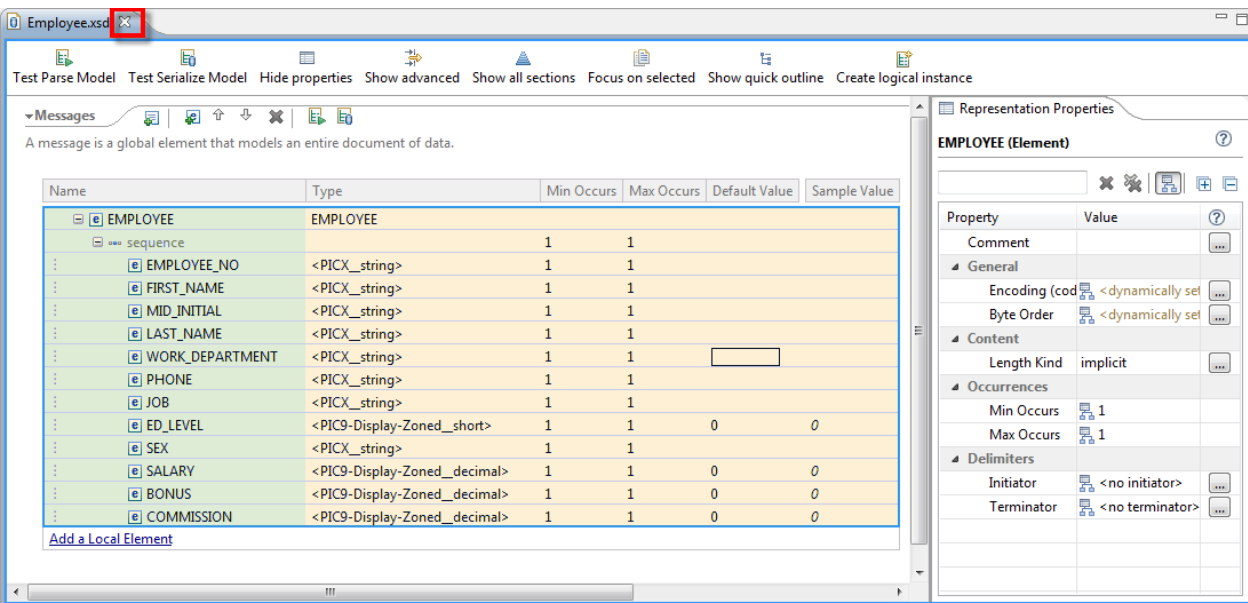
NSYMBOL: ☒ NATIONAL ☐ DBCS

Restore Defaults

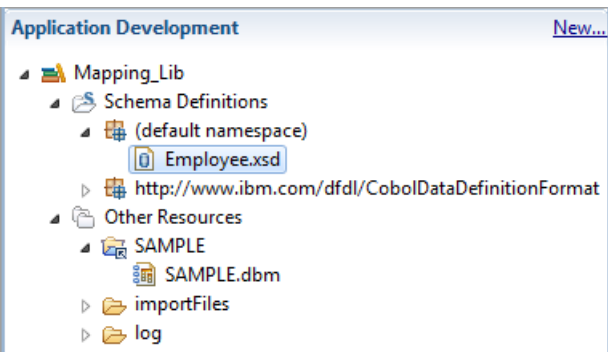
< Back Next > **Finish** Cancel

__11. The new Message Model has now been created, and will look something like the screen capture below. Note that DFDL uses the standard XSD model, so the created file is called Employee.xsd. Note that all hyphens in the COBOL copybook have been converted to underscores in the schema.

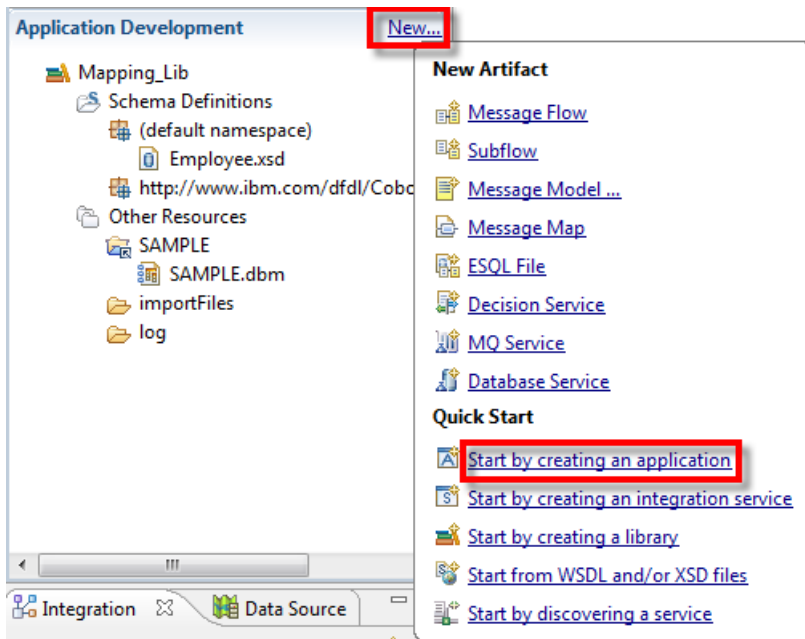
You can explore the schema by clicking on various elements in the left pane; element details will appear in the right pane. Further information on the DFDL Message Modeling tools is provided in other lab guides. When you have finished looking at the Message Model, **close** the schema editor.



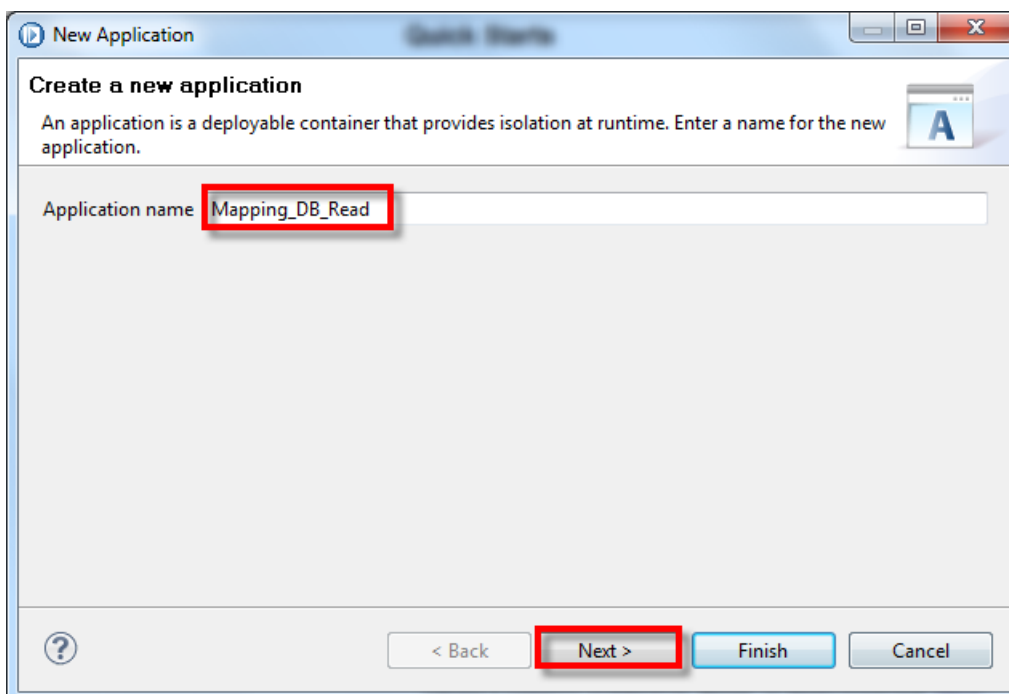
__12. The navigator will now show the complete library, with the database and Message Model schemas.



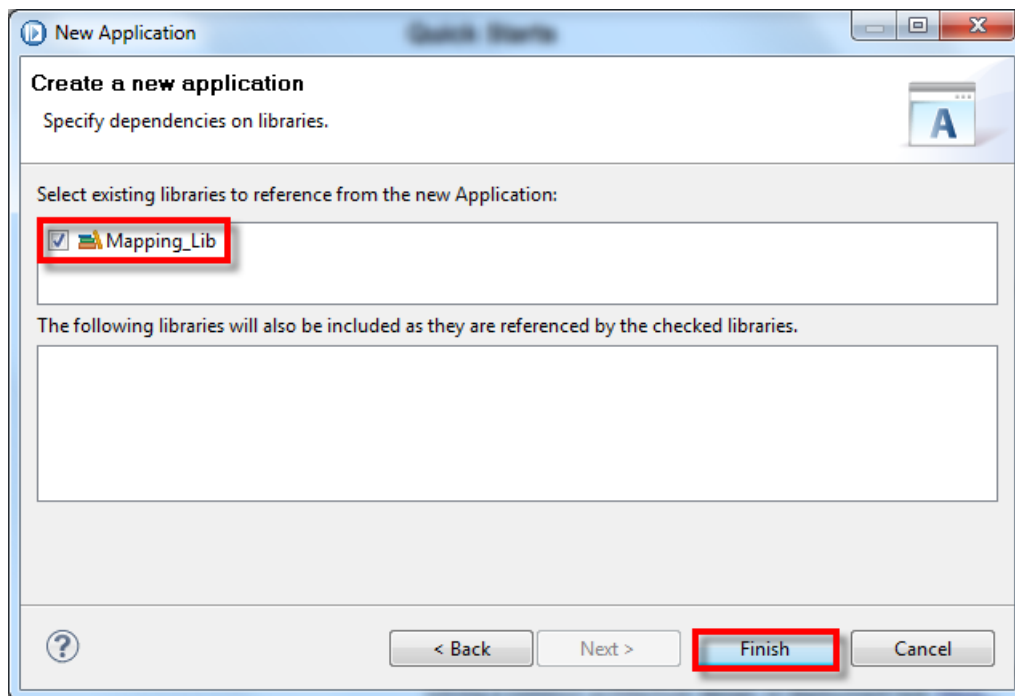
__13. Create a new Integration Bus Application by clicking on the **New** link, and then select “**Start by creating an application**”.



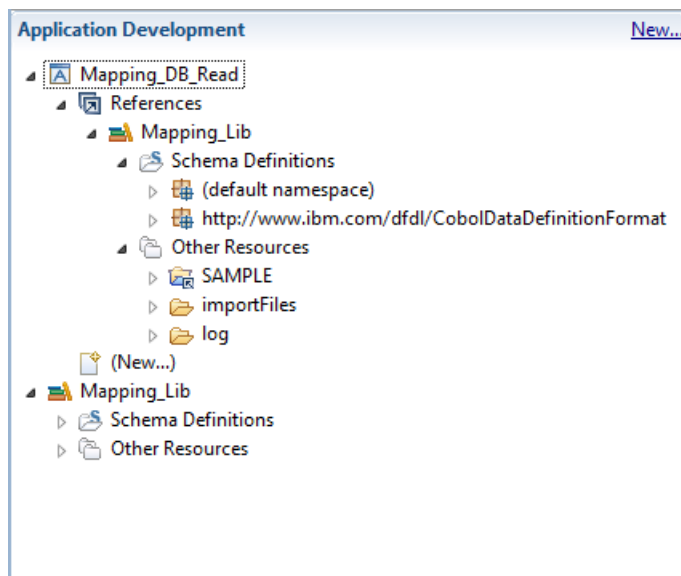
__14. Name the new application **Mapping_DB_Read** and click **Next**.



__15. Check the **Mapping_Lib** library to reference from the new Application. Click **Finish**.



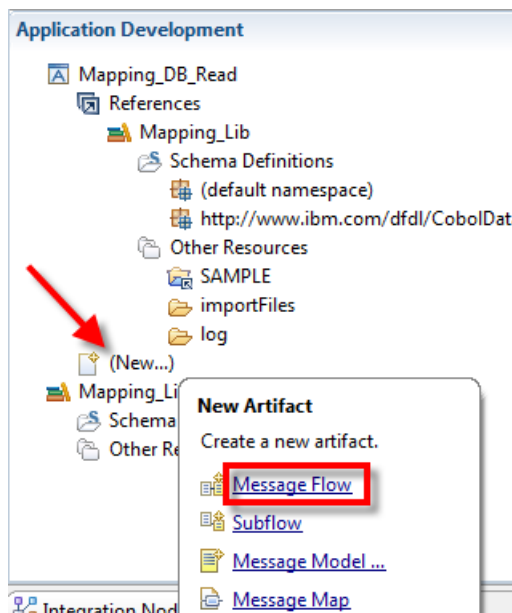
__16. This will result in the following:



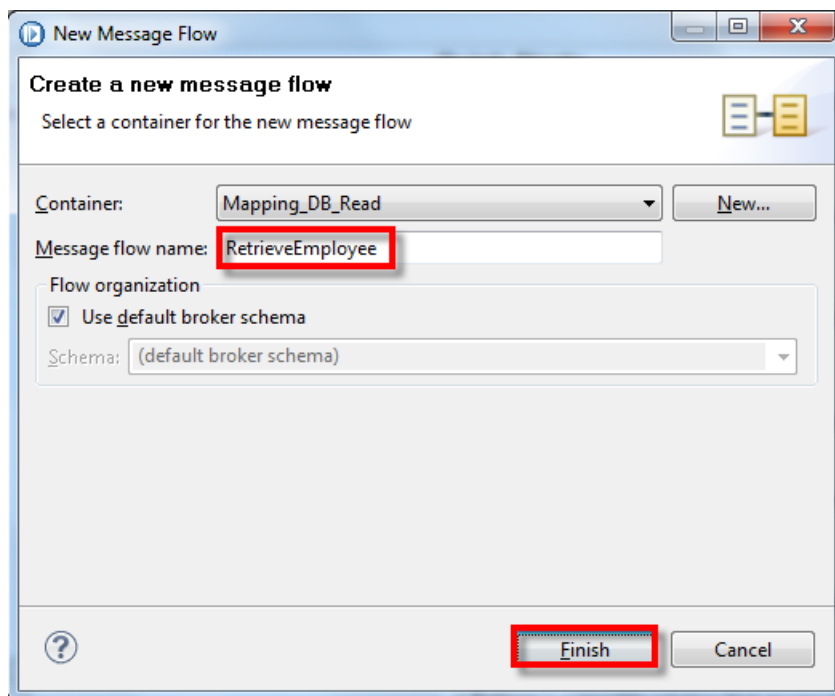
We will now create the message flow which will use the database definition and the Message Model.

We will create a simple flow with just an MQInput, MQOutput and a Mapping Node.

__17. Click the **New** link under the **Mapping_DB_Read** Application and select **Message Flow**.

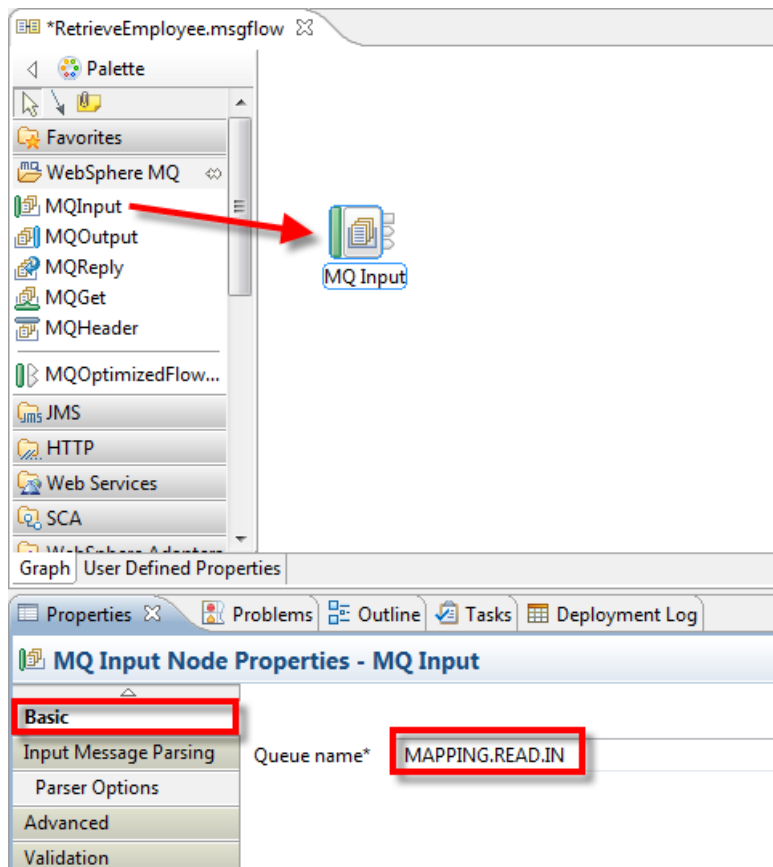


__18. Name the message flow **RetrieveEmployee** and click **Finish**.

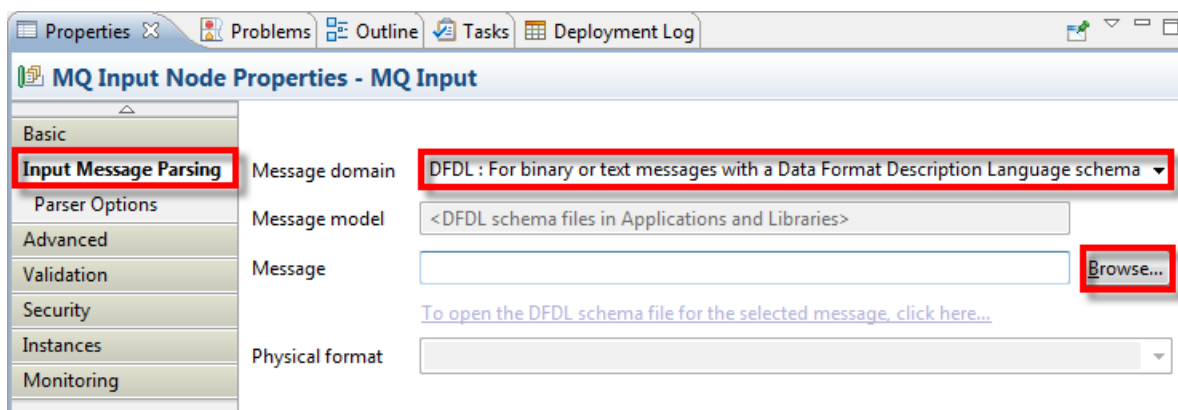


___19. Drop an **MQ Input** node onto the flow editor.

On the **Basic** tab, set the MQ **Queue Name** to **MAPPING.READ.IN**

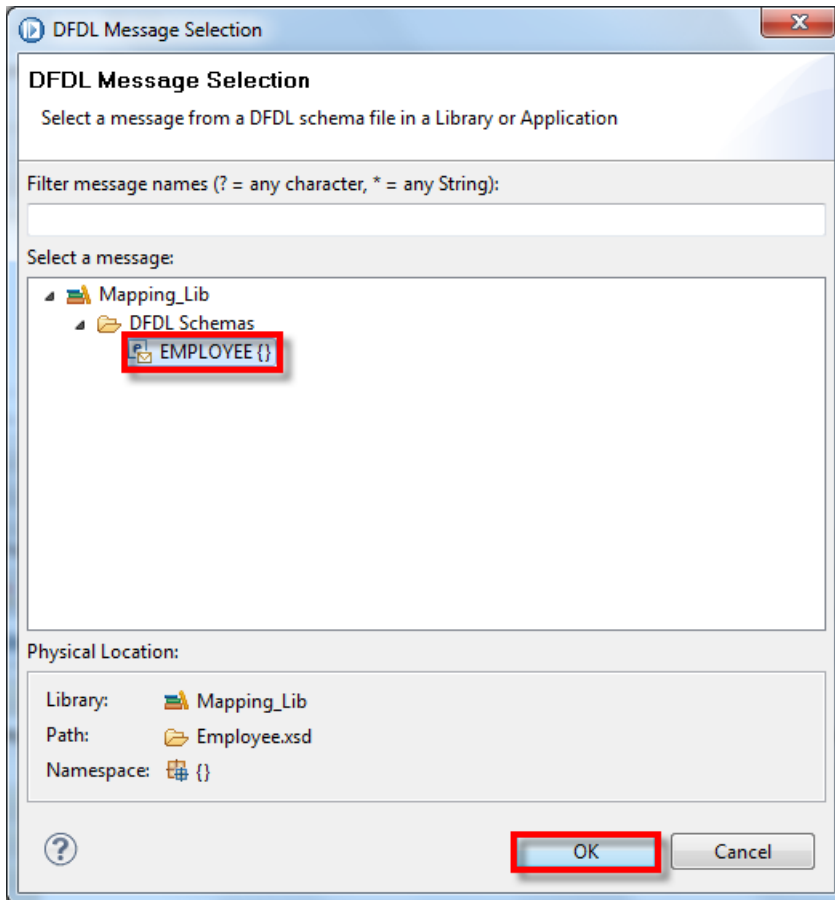


___20. On the **Input Message Parsing** tab, use the drop-down list to set the **Message domain** to **DFDL**. Click the **Browse...** button next to **Message**.

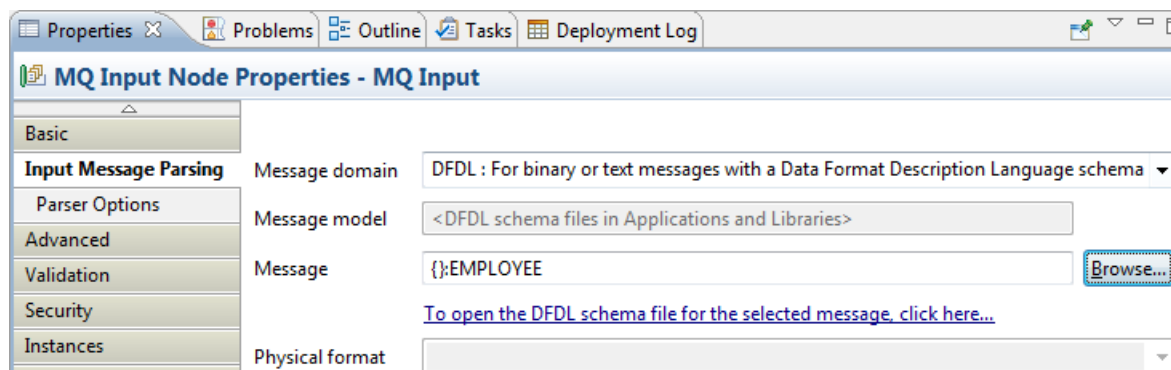


Note for the DFDL domain, the Message Model is greyed out. In this case, the message we select will be available in the DFDL Schemas folder of the Mapping_Lib library. This is available to this Application because of the library reference.

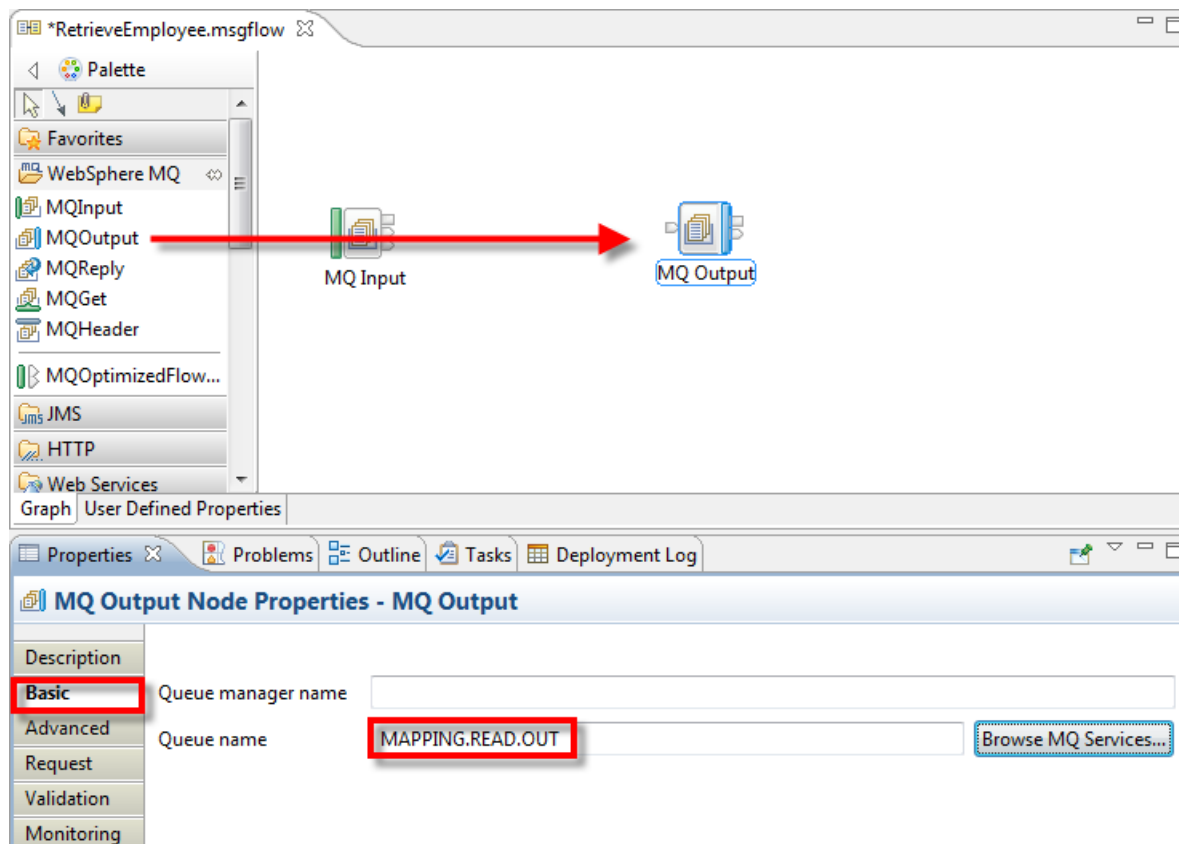
21. Select the **EMPLOYEE** message and click **OK**.



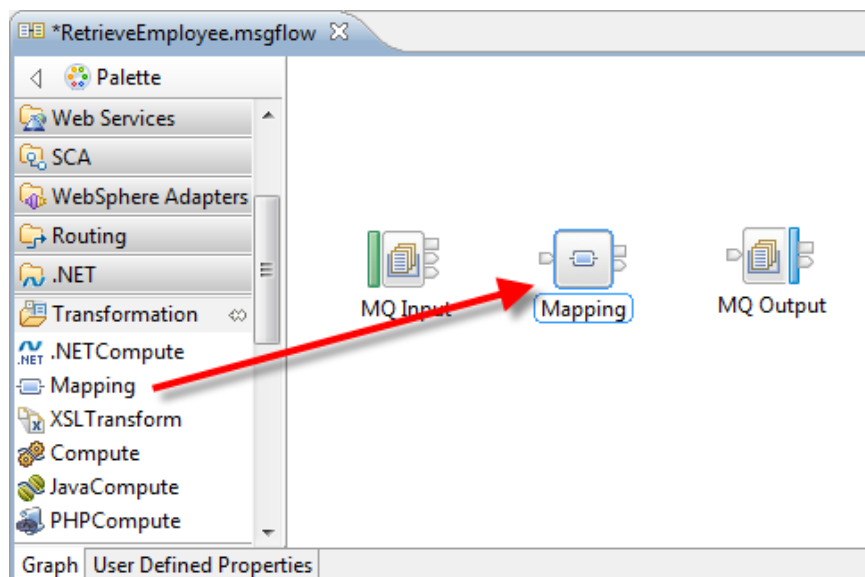
All other properties can remain as default values.



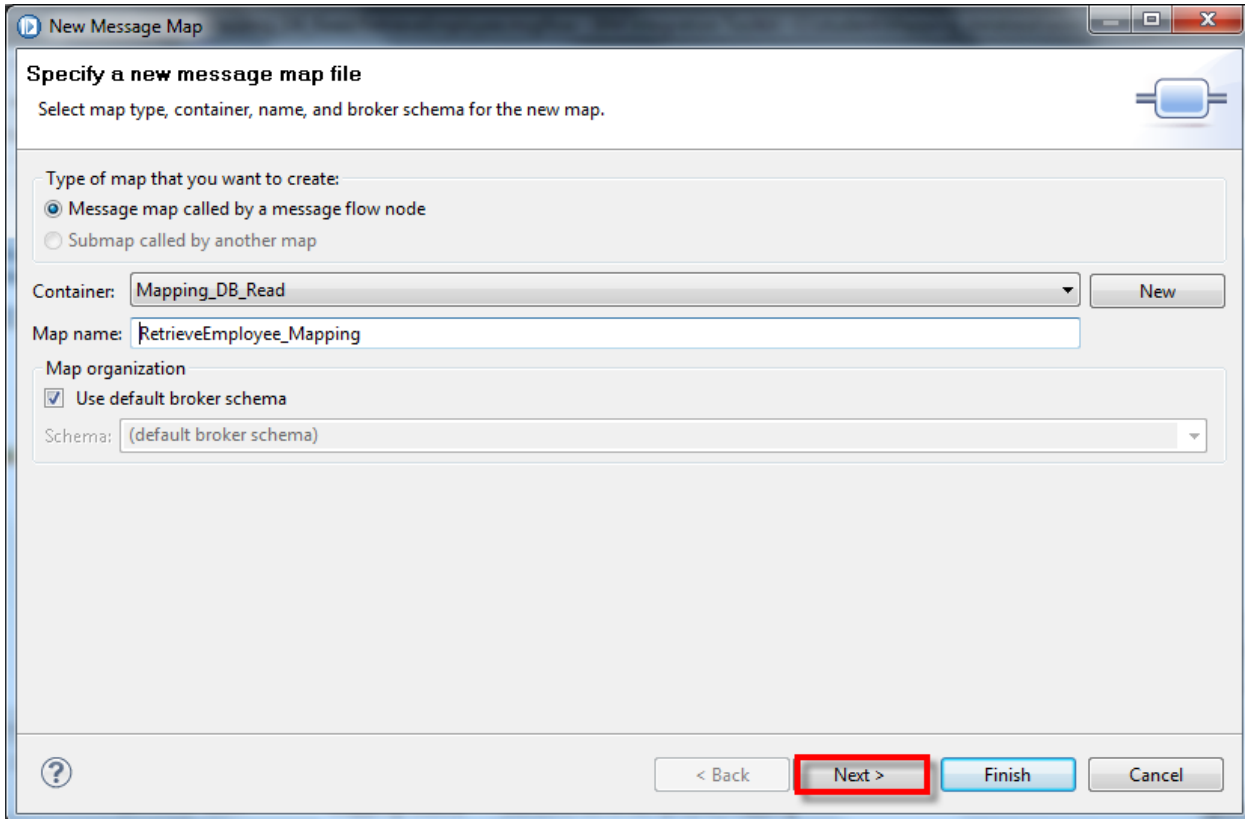
__22. Drop an **MQ Output** node onto the flow editor. Set the **Queue name** to **MAPPING.READ.OUT**



__23. From the Transformation drawer, drop a **Mapping** node onto the flow editor between the MQ Input and MQ Output nodes.



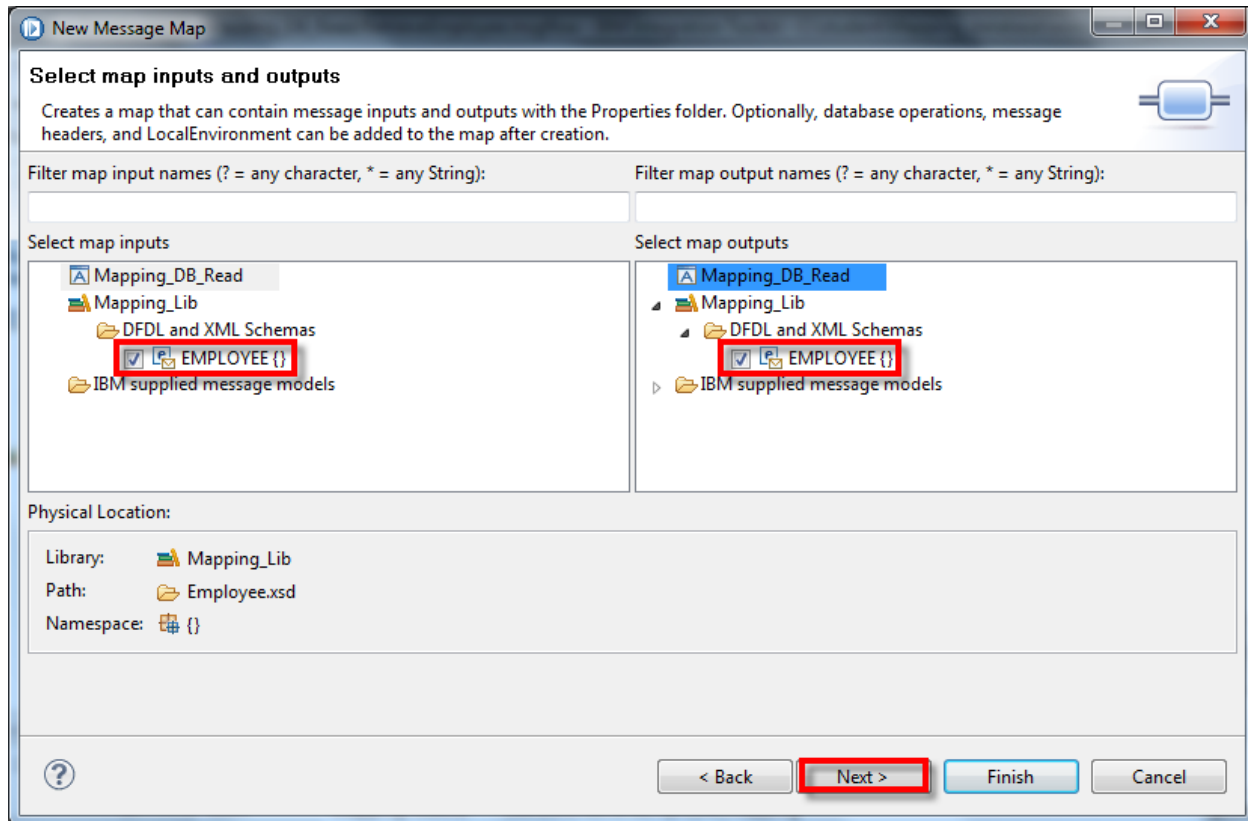
24. Double-click the **Mapping** node to open the Mapping wizard. The first screen specifies the Application name and map name; accept the defaults and click **Next**.



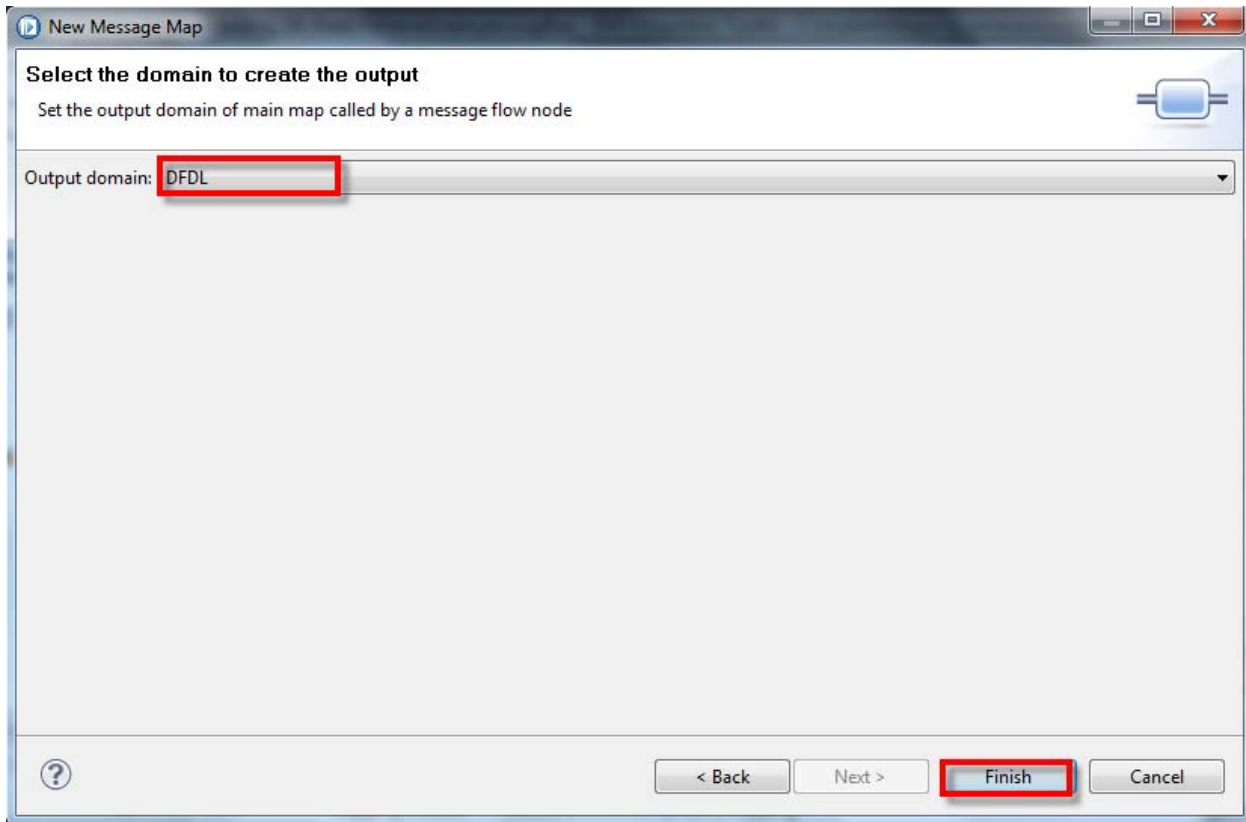
The image shows a 'New Message Map' dialog box with the following fields and options:

- Title:** New Message Map
- Instruction:** Specify a new message map file. Select map type, container, name, and broker schema for the new map.
- Type of map that you want to create:**
 - ☒ Message map called by a message flow node
 - ☐ Submap called by another map
- Container:** Mapping_DB_Read (dropdown menu)
- Map name:** RetrieveEmployee_Mapping (text input)
- Map organization:**
 - ☒ Use default broker schema
- Schema:** (default broker schema) (dropdown menu)
- Buttons:** < Back, Next > (highlighted with a red box), Finish, Cancel

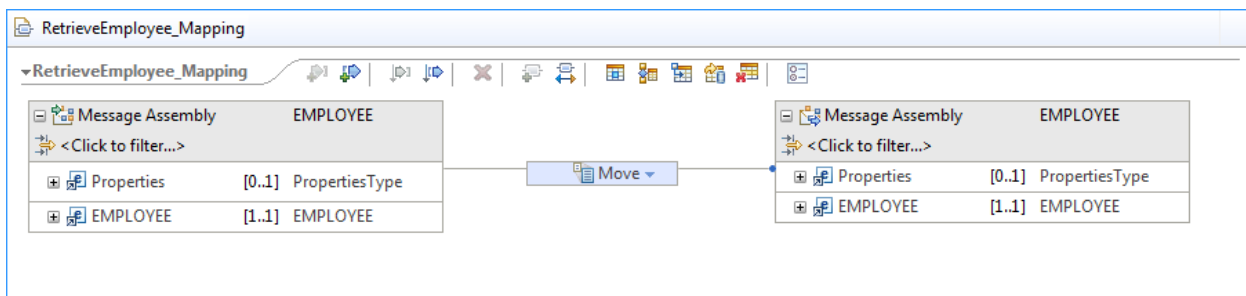
___25. Expand the available map inputs and outputs, and select **EMPLOYEE** from the Mapping_Lib library for both the input and output. Click **Next**.



__26. Select the **Output domain** to **DFDL**. Click **Finish**.

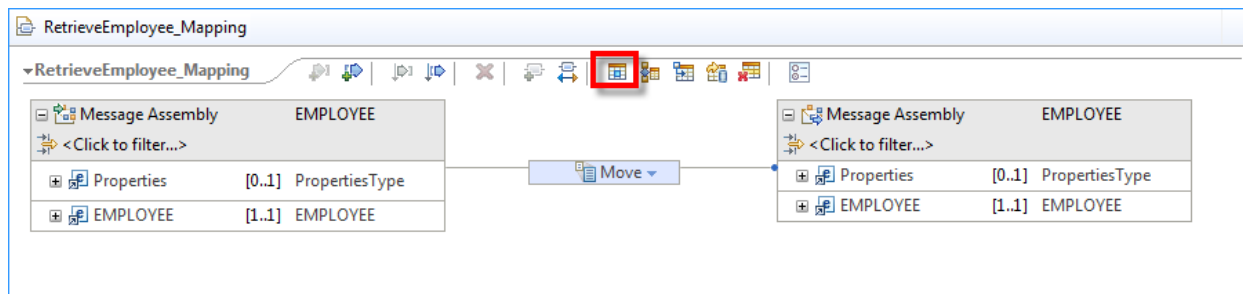


__27. The Map Editor will now open. The basic inputs and outputs are available (based on the Employee.xsd Message Model). The message properties have been mapped automatically, but no other mappings have been done at this stage.



__28. We now need to add a database selection input, so that we can retrieve employee information.

Click on the **Database** icon on the top row of the map editor, as shown. If you are not sure which icon it is, hover over it, and it will show the text “Select rows from a database”.



This will open the “New Database Select” window.

__29. The Database Select window is the primary tool that is used to tell the Mapping node which database table to access, and what query to perform against the selected table.

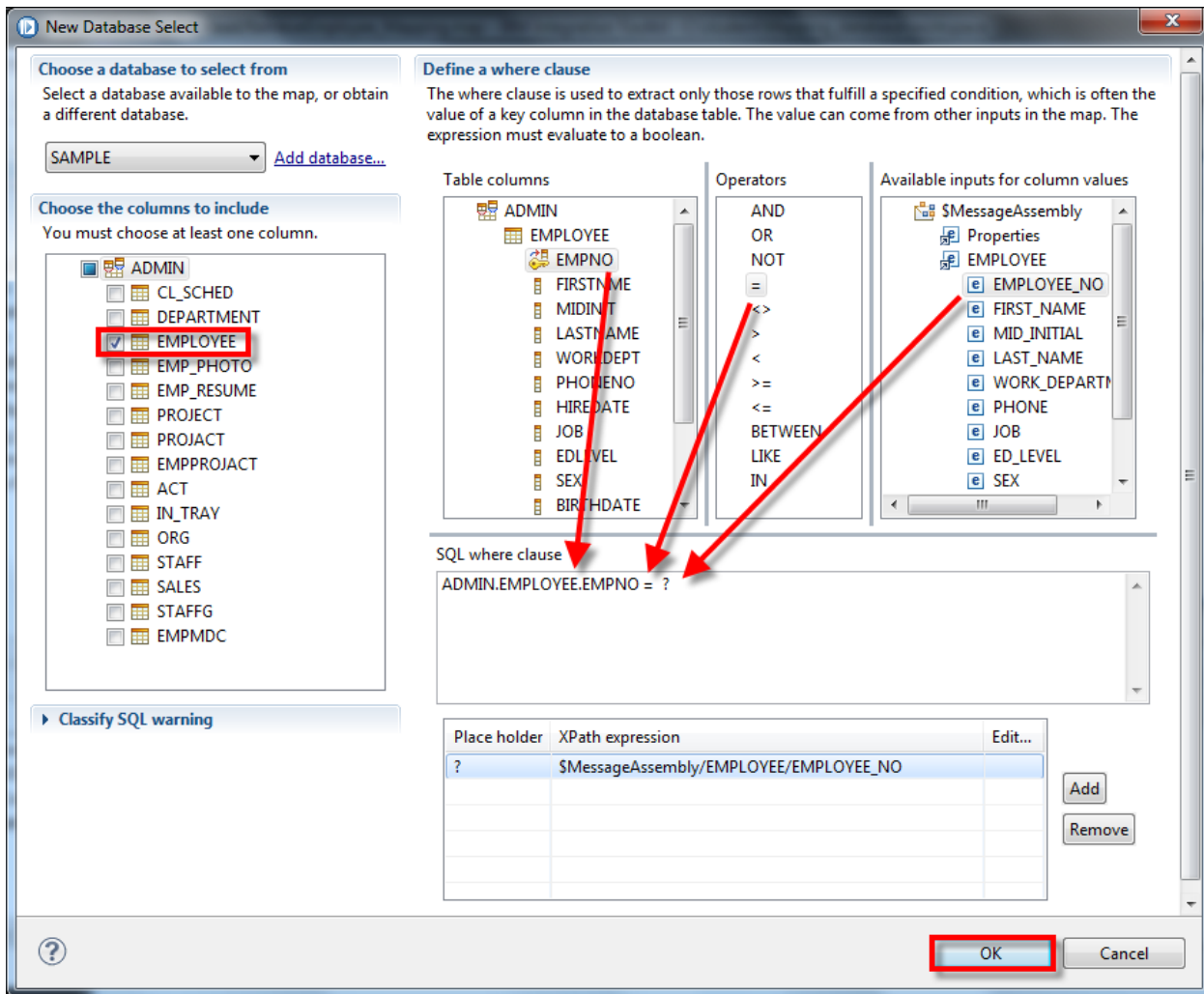
In this window, make the following selections:

- i. In the left-hand pane (**Choose the columns to include**), select the **EMPLOYEE** table. If you expand the **EMPLOYEE** table, you will see that all columns have been automatically selected. You can deselect any of these if you wish, but leave them all selected for this example.
- ii. In the “**SQL where clause**” pane, remove the default contents “**1=1**”.
- iii. From the **Table columns** pane, drag and drop the **EMPNO** column onto the “**SQL where clause**”.
- iv. From the **Operations** pane, drop an “**=**” onto the “**SQL where clause**” (or you can type it manually).
- v. From the “**Available inputs**” pane, expand the **EMPLOYEE** item, and drop the **EMPLOYEE_NO** onto the “**SQL where clause**”. Note that this action will generate the SQL clause “EMPLOYEE.EMPNO=?”. The “?” is a reference to the fully qualified value shown in the field below as an XPath expression:

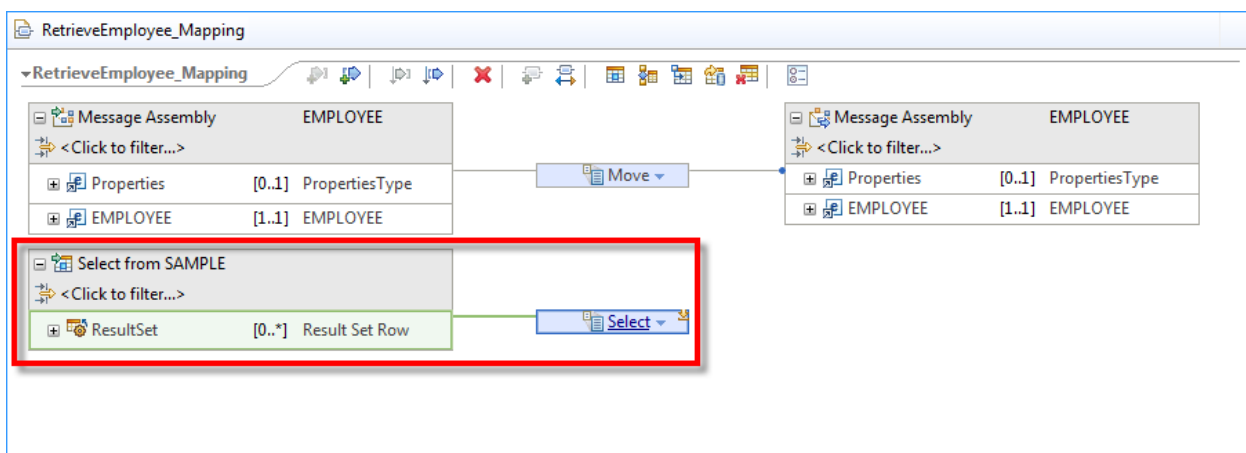
\$MessageAssembly/EMPLOYEE/EMPLOYEE_NO

Alternatively, in more complex scenarios, you can construct your own XPath expressions in this field.

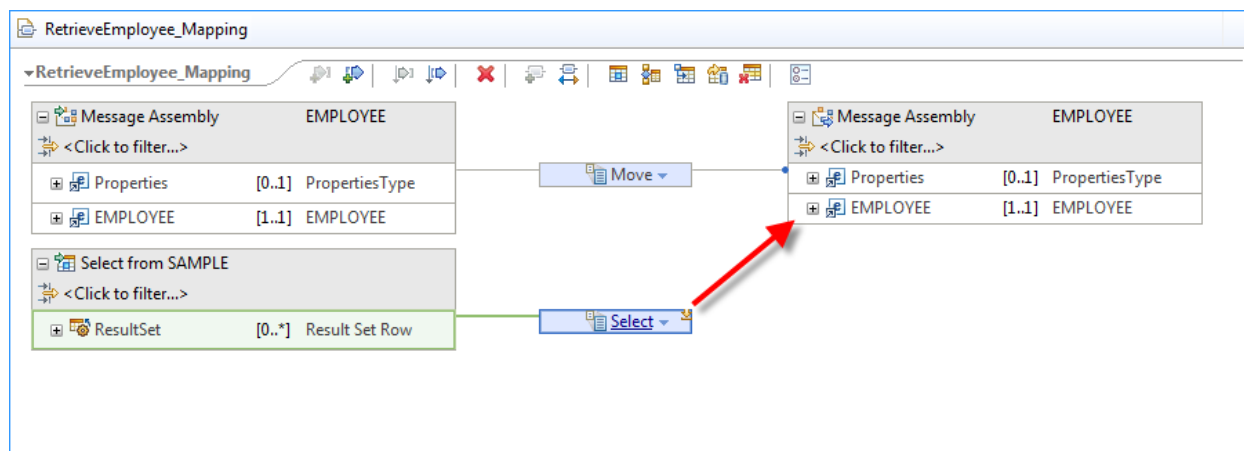
- vi. Click **OK**.



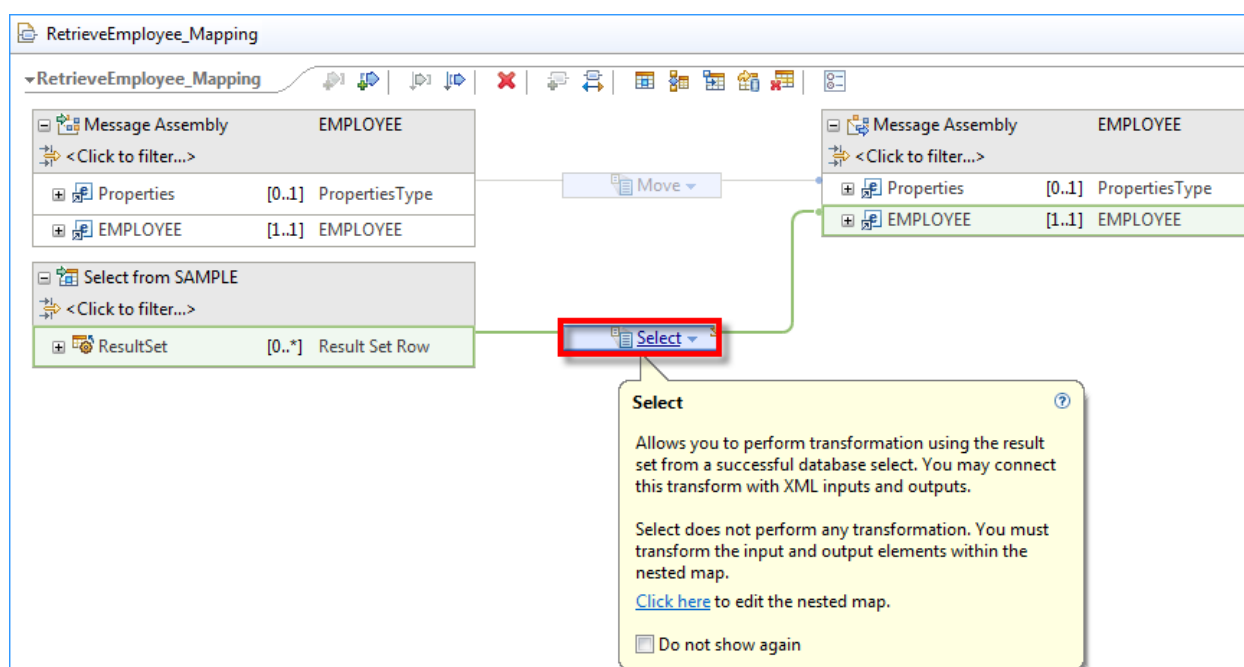
__30. We have now added the input selection for the SAMPLE database, as highlighted below.



__31. Note that adding the database input selection has also created a default mapping function for us. To complete the selection, highlight the **"Select"** map icon, and drag and drop it onto the **EMPLOYEE** output message.



This results in the following mapping.



__32. To create the precise mappings that we need, click the **“Select”** transform name on the map icon. This will open a local map which will contain details of all available fields for input and output.

Click the **Auto Map** button to automatically generate the required mappings. Note the element names are quite different from the database EMPLOYEE table (the input) to those generated from the COBOL copybook (the output).

(Hover over the icon, and it will show the text “Automap input to output”).

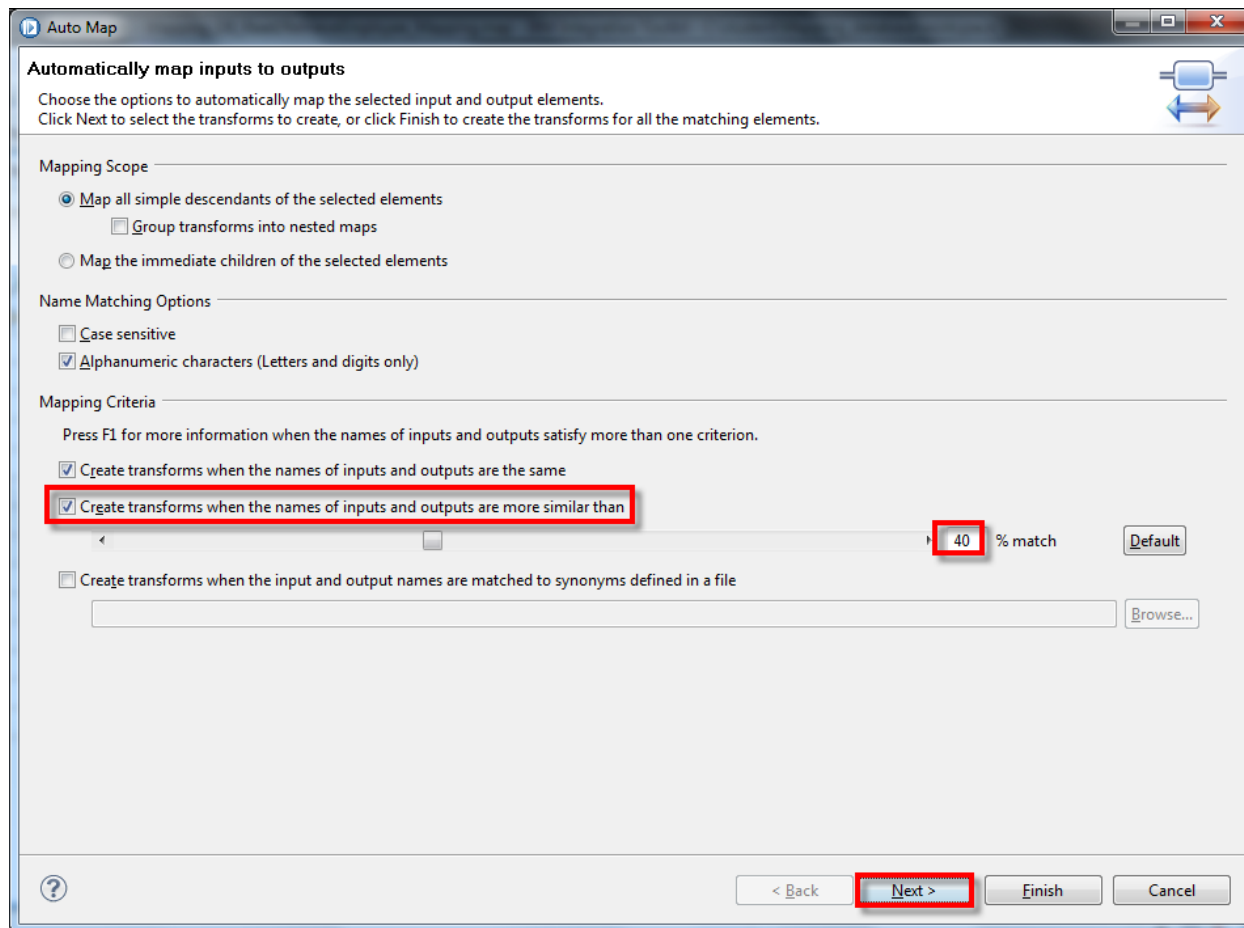
The screenshot shows the IBM Data Mapper interface for a map named 'RetrieveEmployee_Mapping'. The 'Auto Map' button, represented by a blue icon with a double-headed arrow, is highlighted with a red box in the toolbar. Below the toolbar, two data tables are displayed side-by-side. The left table, titled 'ResultSet', lists database fields: EMPNO, FIRSTNAME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, and COMM. The right table, titled 'EMPLOYEE', lists COBOL copybook fields: EMPLOYEE_NO, FIRST_NAME, MID_INITIAL, LAST_NAME, WORK_DEPARTMENT, PHONE, JOB, ED_LEVEL, SEX, SALARY, BONUS, and COMMISSION. The 'SALARY' field in the 'EMPLOYEE' table is highlighted with a yellow box, and a yellow line connects it to the 'SALARY' field in the 'ResultSet' table.

ResultSet		EMPLOYEE	
EMPNO	[1..1] CHAR	EMPLOYEE_NO	[1..1] <PICX_string>
FIRSTNAME	[1..1] VARCHAR	FIRST_NAME	[1..1] <PICX_string>
MIDINIT	[1..1] CHAR	MID_INITIAL	[1..1] <PICX_string>
LASTNAME	[1..1] VARCHAR	LAST_NAME	[1..1] <PICX_string>
WORKDEPT	[1..1] CHAR	WORK_DEPARTMENT	[1..1] <PICX_string>
PHONENO	[1..1] CHAR	PHONE	[1..1] <PICX_string>
HIREDATE	[1..1] DATE	JOB	[1..1] <PICX_string>
JOB	[1..1] CHAR	ED_LEVEL	[1..1] <PIC9-Display-Zoned_short>
EDLEVEL	[1..1] SMALLINT	SEX	[1..1] <PICX_string>
SEX	[1..1] CHAR	SALARY	[1..1] <PIC9-Display-Zoned_decimal>
BIRTHDATE	[1..1] DATE	BONUS	[1..1] <PIC9-Display-Zoned_decimal>
SALARY	[1..1] DECIMAL	COMMISSION	[1..1] <PIC9-Display-Zoned_decimal>
BONUS	[1..1] DECIMAL		
COMM	[1..1] DECIMAL		

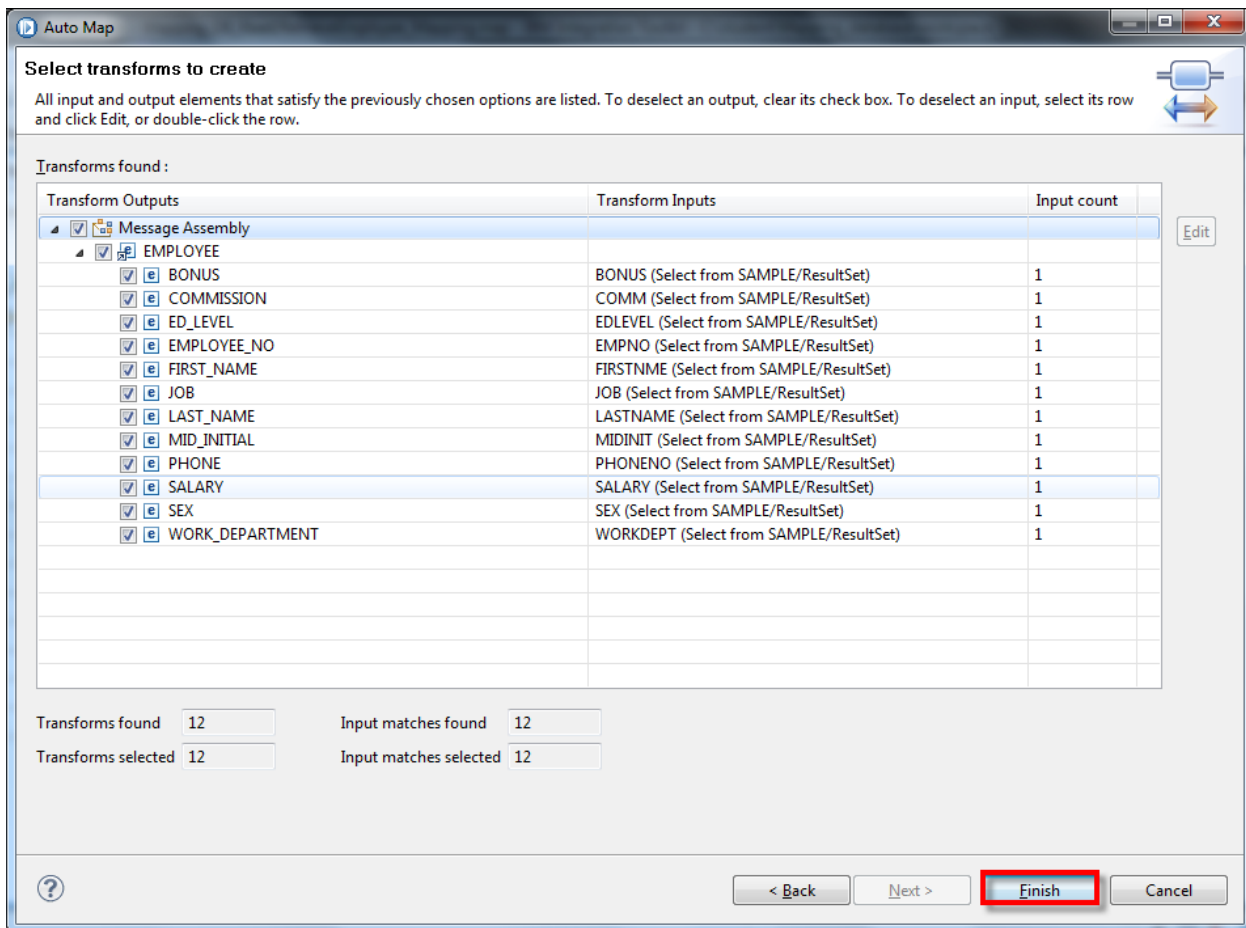
__33. On the Auto Map window, check the “**Create transforms when the names of inputs and outputs are more similar than**” checkbox.

Set the percentage match to **40%** (this is required because the input and output field names are significantly different).

Click **Next**.

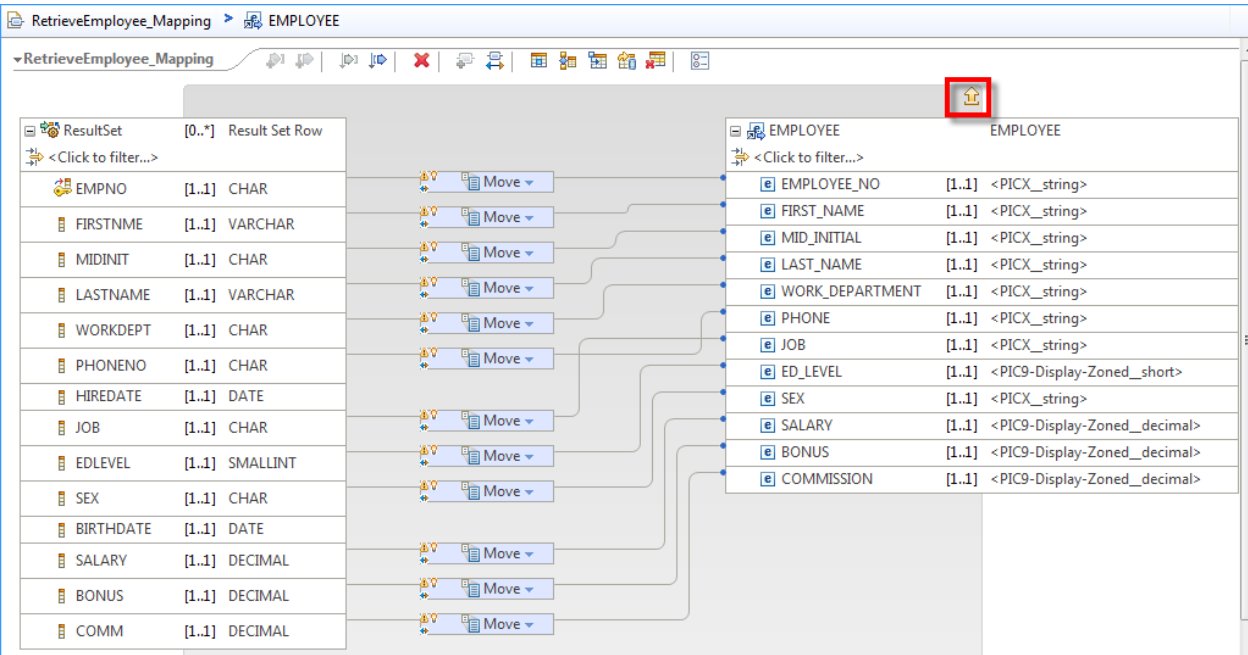


34. The next window will show you the field mappings that have been automatically created, using the 40% similarity figure. Note that all fields have been mapped correctly. Click **Finish**.

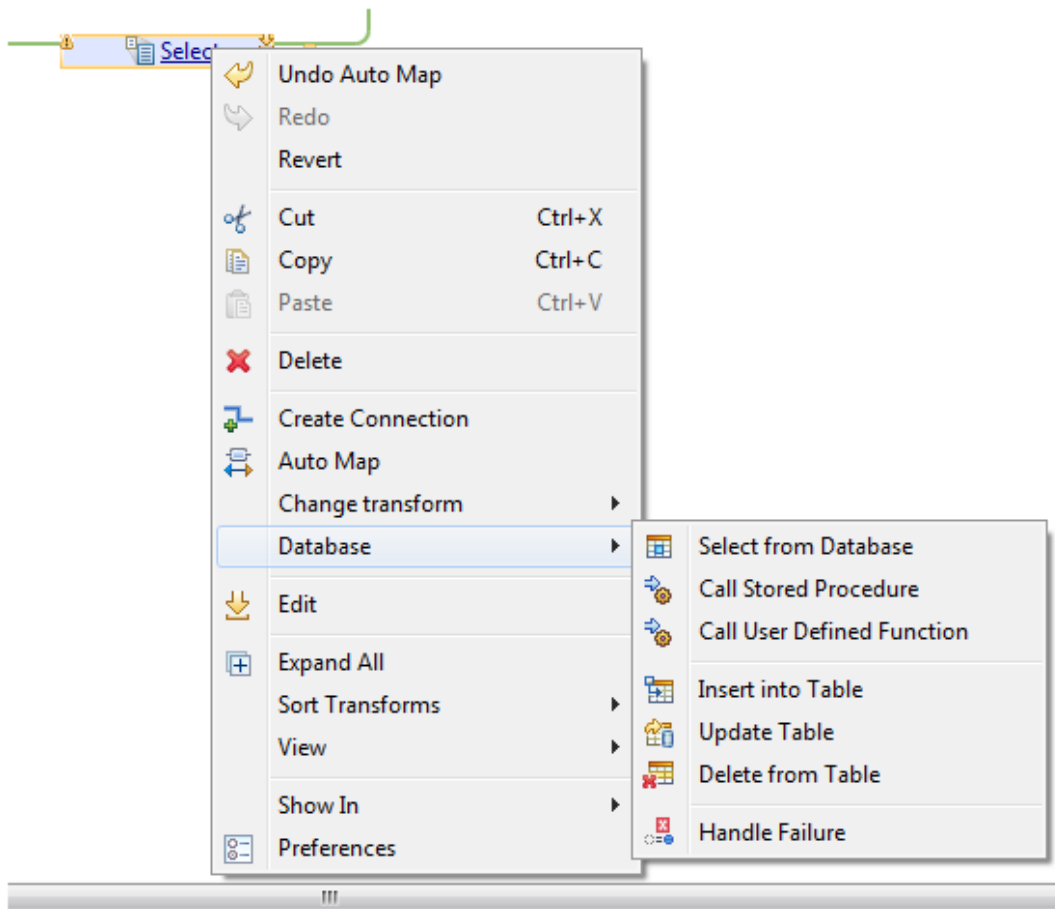


35. The local map has now been completed, and all input fields from the database result set have been mapped onto the output EMPLOYEE record.

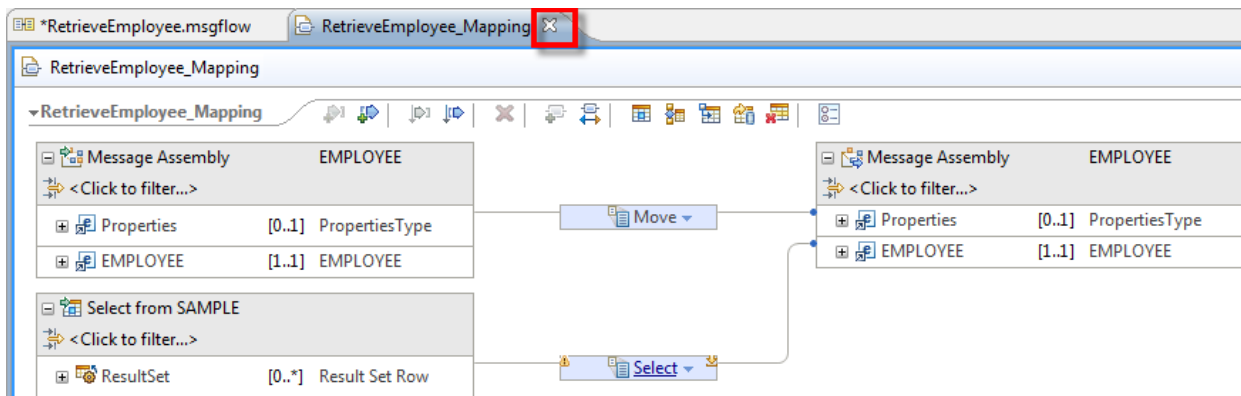
Click “Up a level” (the **yellow arrow** in the top right).



__36. Please note that here we can add a map function to handle database failure by right clicking on 'Select' and choosing 'Handle Failure'. The screen shot below is given for reference only.



__37. Save (Ctrl+S) and close the map.

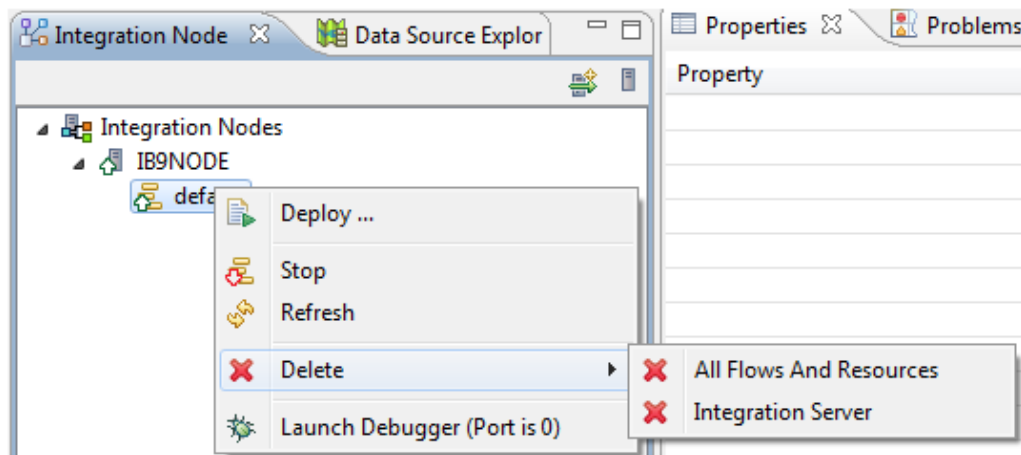


__38. Finally, connect the nodes as shown, and save the flow.

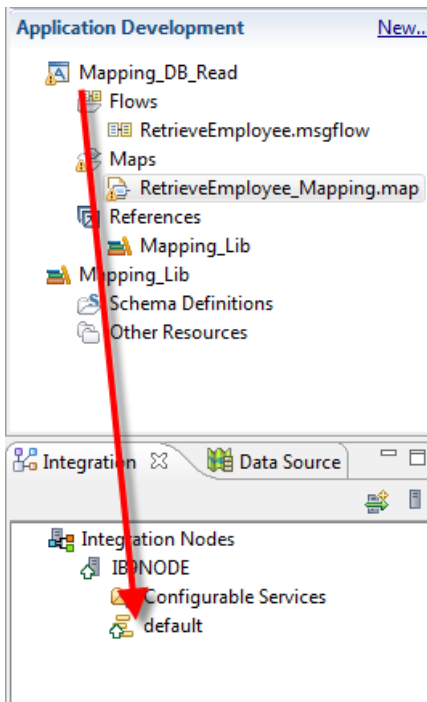


3.3.1 Deploy and test the application

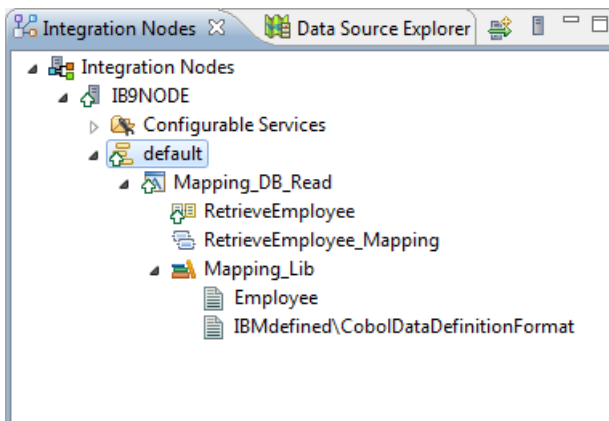
__1. First, to ensure your new application does not interfere with any other deployed application, delete all deployed flows from the default Integration server. In the Integration Node view, right-click the **default** Integration Server and select **Delete All Flows and Resources**.



__2. In the Applications view, highlight the **Mapping_DB_Read** application, and drag and drop it onto the **default** Integration Server (or right-click the application and select **Deploy** from the context menu).



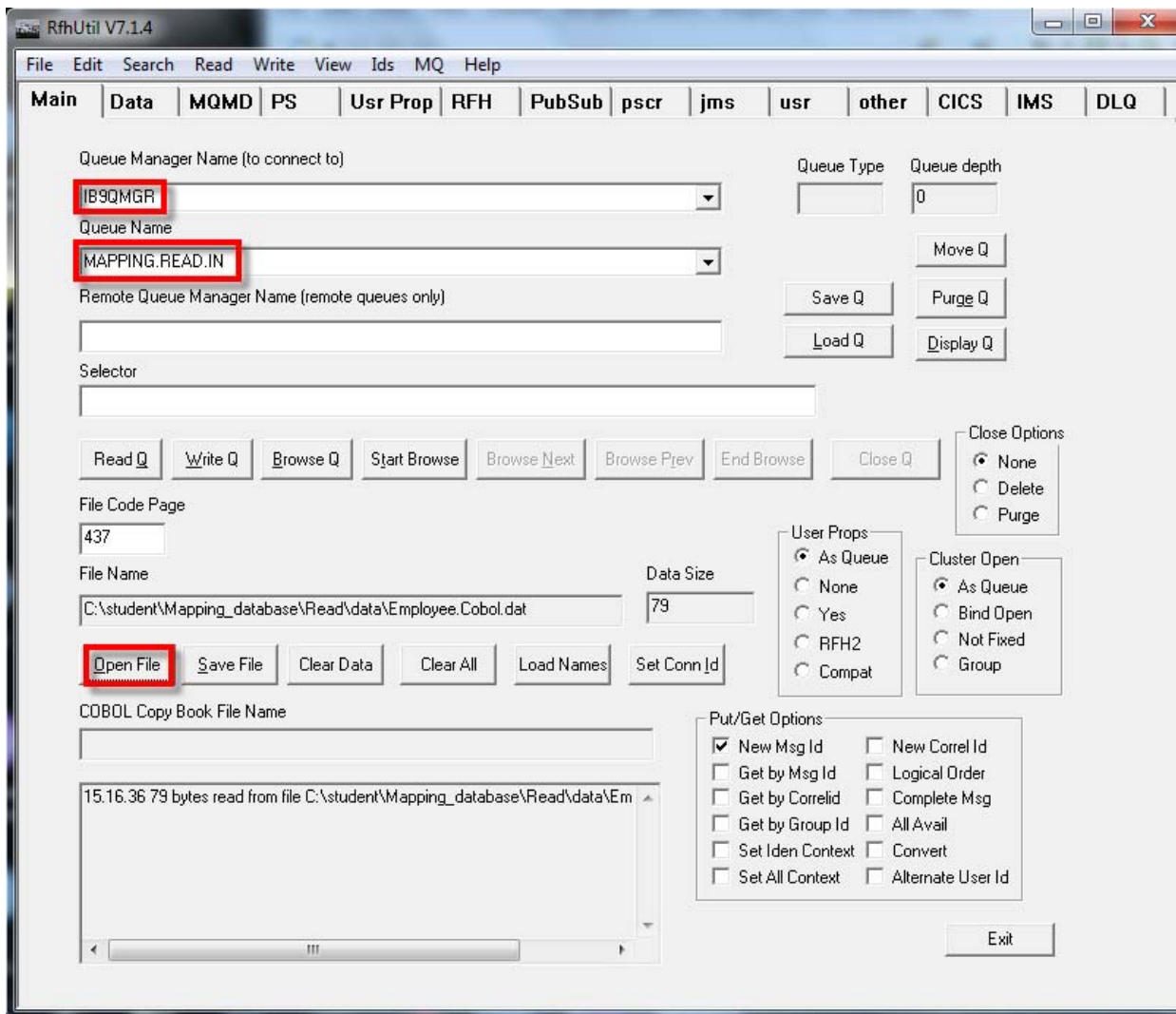
__3. The deployed application will look like this. Note that the Mapping_DB_Read application contains the RetrieveEmployee message flow and the RetrieveEmployee_Mapping map. It also contains the library Mapping_Lib, which contains the required message and database schemas.



__4. In this example, we will use RFHUtil to send an MQ message to the application. Open an instance of **RFHUtil** from the desktop, use the drop-down boxes to set the Queue Manager Name to **IIB9QMGR**, and set the Queue name to **MAPPING.READ.IN** using the drop-down box.

Click **Open File**, and open the file

c:\student \ mapping_database \ Read \ data \ Employee.Cobol.dat.



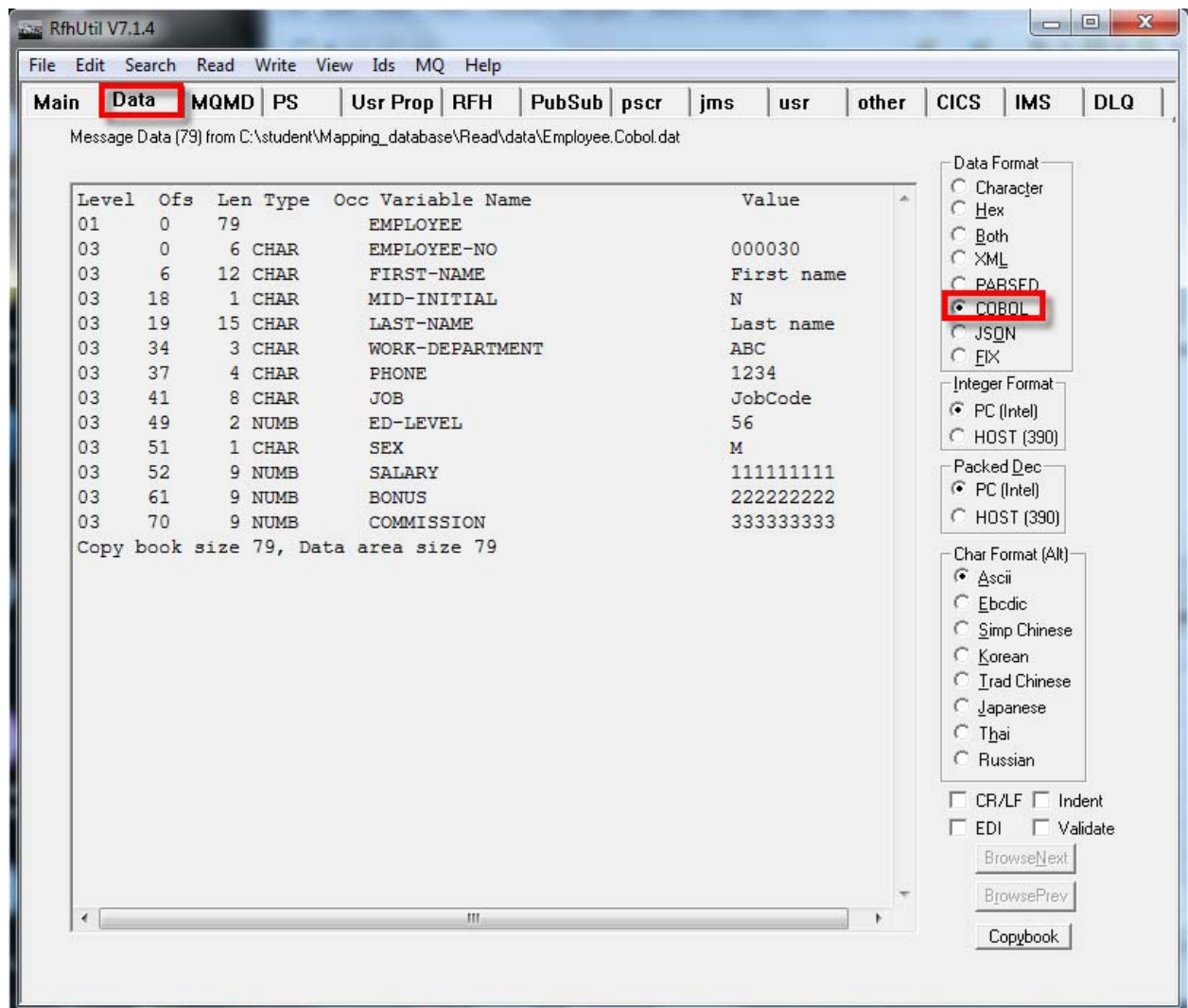
__5. Click the **Data** tab to take a look at the data that we will send in to the application. Initially, the data will appear in its raw format. Click the **COBOL** radio button to apply the original COBOL copybook to this data. As a reminder, the copybook is stored in:

c:\student\mapping_database\resources\Employee.cpy.

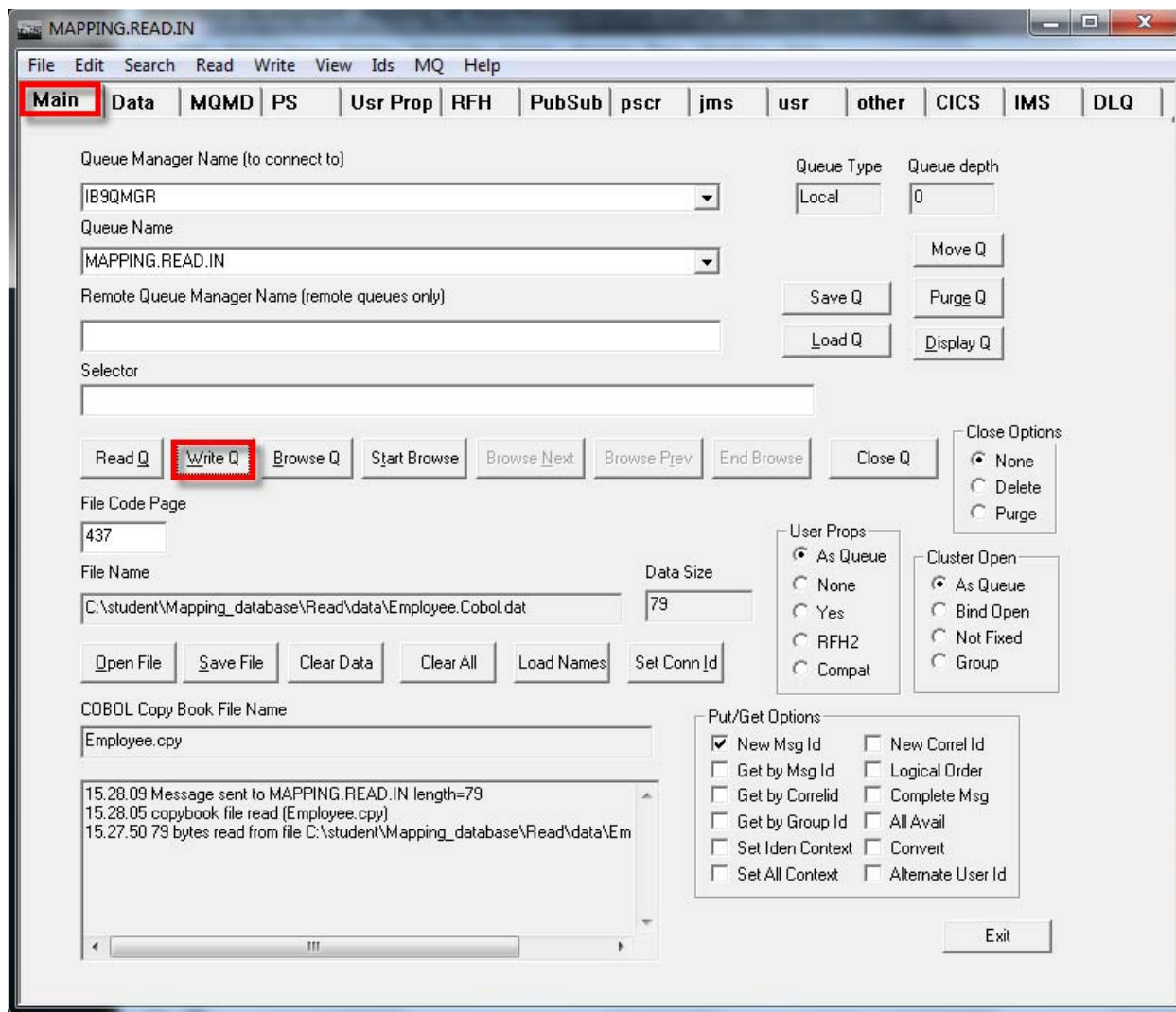
Note that the data contains a value for EMPLOYEE-NO. The value shown below is 000030.

The input file may have been changed for other scenarios, but this value should be one of 000010, 000020, 000030.

Note that the other data fields have just been populated with dummy data.

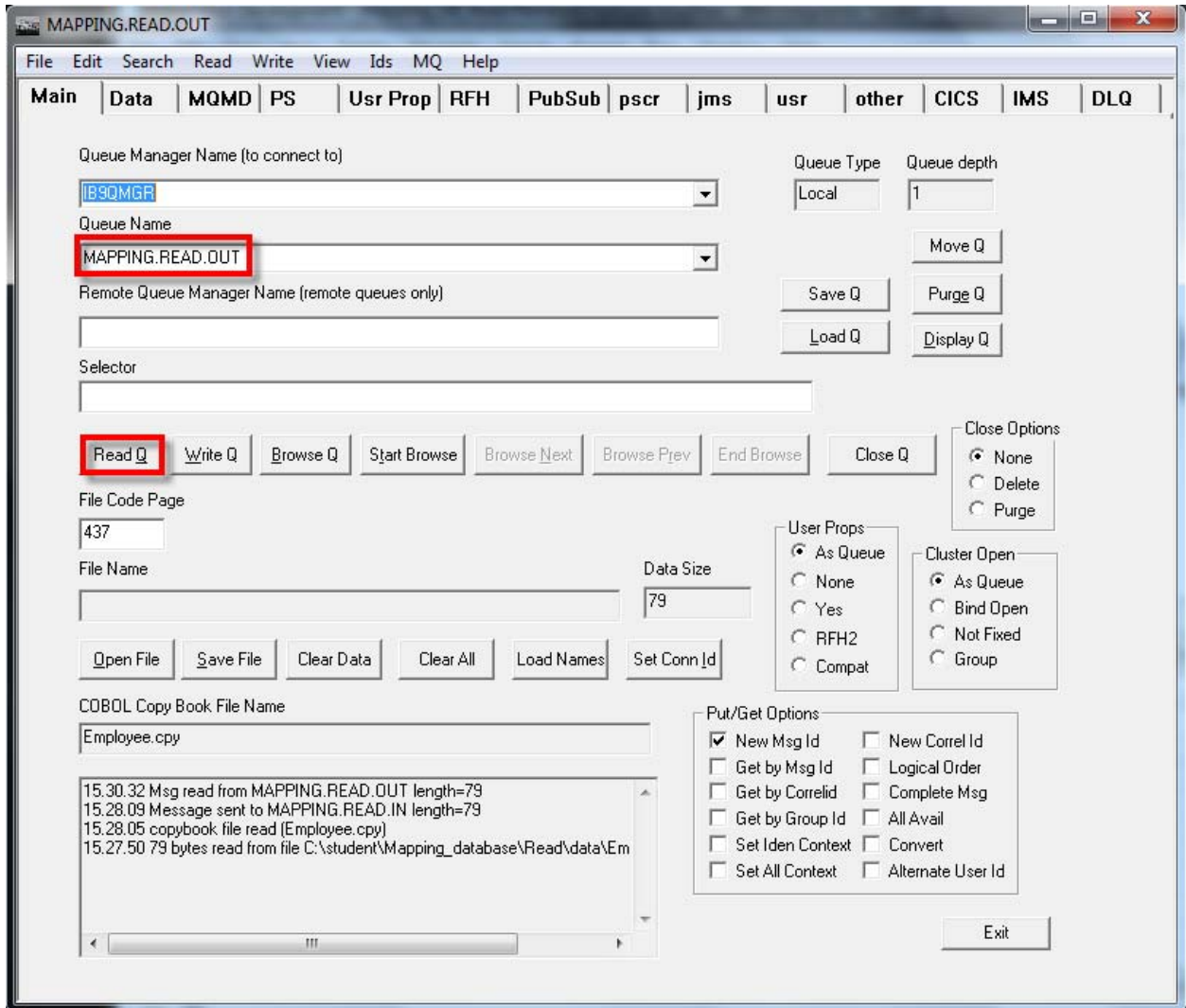


__6. Click back to the **Main** tab, then click **Write Q**.



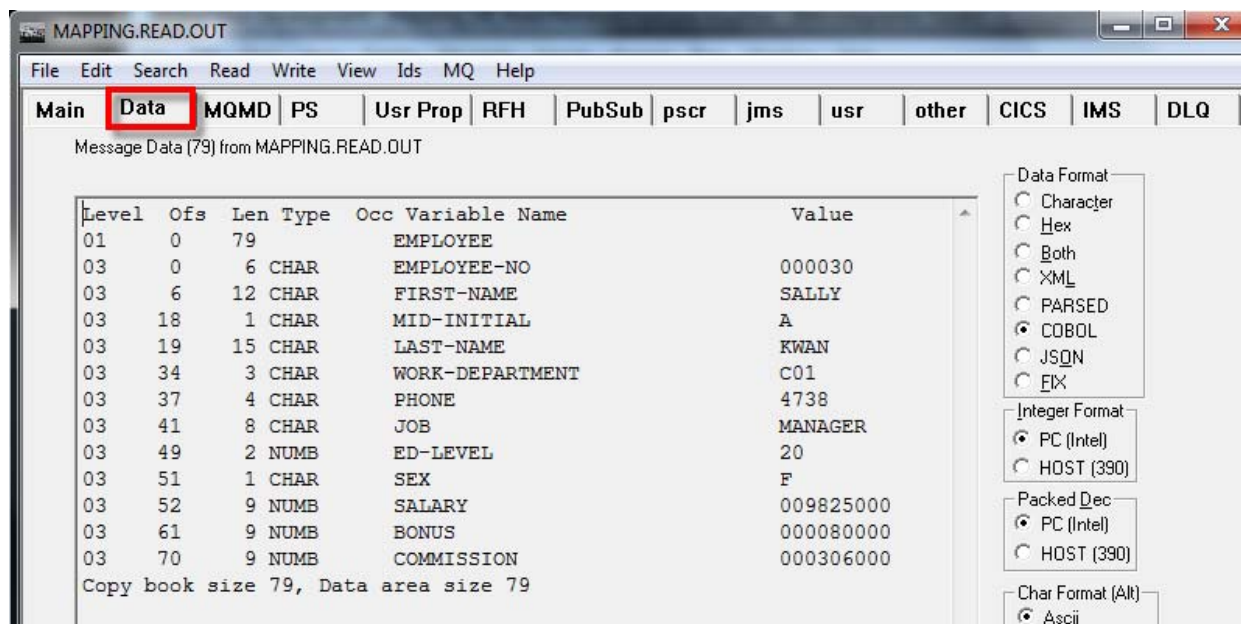
__7. The application will now process the message that you have just sent.

On the Main tab, use the drop-down to set the **Queue Name** to **MAPPING.READ.OUT** and then click **Read Q**.

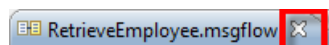


__8. After RFHUtil successfully reads a message (you may need to click ReadQ a few times, since the application will need to connect to the database, which may also be paged out of memory), click the **Data** tab.

Note that the required database row from the EMPLOYEE table has been retrieved and has been mapped to the format required by the original COBOL copybook.



__9. Close RFHUTIL and close the **RetrieveEmployee.msgflow**.



This concludes the basic introduction to the Mapping node and database retrievals.

__10. Proceed with **Section 3.5** to clean up if you are not doing the next section.

3.4 Using a Stored Procedure in a map

IBM Integration Bus Version 9 expands the capabilities of the Mapping node to support Database Stored Procedures.

A **Stored Procedure** is simply a program that is stored on the database server. Usually written in SQL, the Stored Procedure benefits from the power and proximity of the database from which it is managed.

Stored Procedures offer you many other benefits, as well.

- They allow you to encapsulate code. In other words, the database operation appears once, in the stored procedure, not multiple times throughout your application source. This improves debugging as well as maintainability.
- Changes to the database schema affect your source code in only one place, the stored procedure. Any schema changes then become a DBA task rather than a wholesale code revision.
- Since the Stored Procedures reside on the server, you can set tighter security restrictions on the client space, saving more trusted database permissions for the well-protected Stored Procedures themselves.
- Since Stored Procedures are compiled and stored outside the application, they can use more sensitive variables within the SQL syntax, such as passwords or personal data that you would avoid using in scripts or remote calls.
- Using Stored Procedures greatly reduces network traffic.

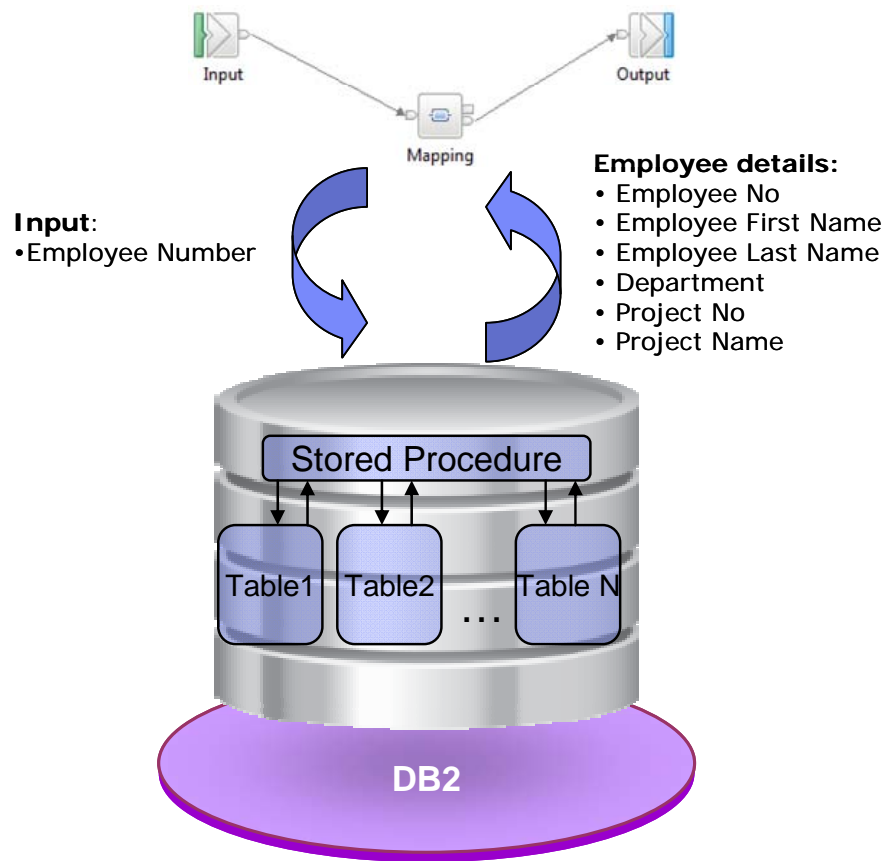
If you just put a series of SQL statements in your client code, each line is executed by sending a message over the network to the server, usually getting a response in return. But a Stored Procedure resides on the server. When called from the client application, it executes on the server and only has to respond when returning the final result set to the client, saving lots of back-and-forth traffic.

Please note that currently, Integration Bus supports only Stored Procedures for DB2. This support will be expanded in the future.

In this lab, you will investigate the new Mapping Node in IBM Integration Bus Version 9 and its support for DB Stored Procedures.

You will first connect to the SAMPLE database and create a new Stored Procedure that will select columns from different tables. You will deploy the procedure to the database and test it. Further, you will import a message flow and you will use the new Stored Procedure for your database mapping. Finally, you will test the message flow in the Integration Bus test tool.

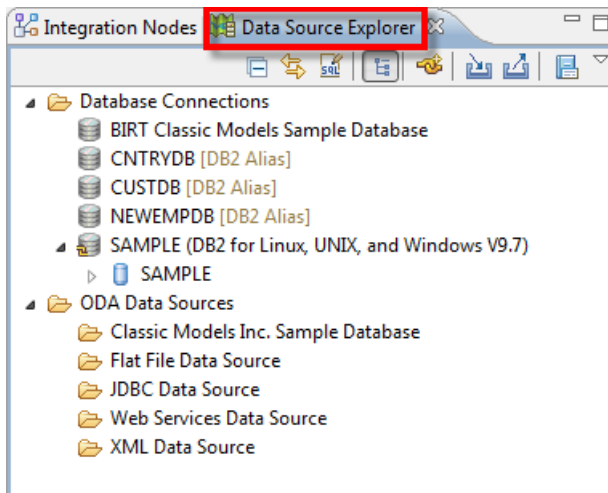
You will be able to experience the benefits of using Stored Procedures to simplify a mapping application when data is required from different tables in a database. Also, you will experience the tools that have been integrated in IBM Integration Bus for creating, deploying and implementing Stored Procedures.



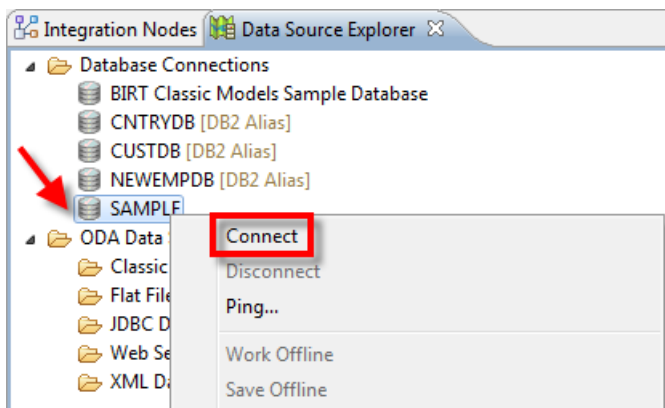
3.4.1 Database definition and DB2 Stored Procedure

Be sure you have completed the steps in **Section 3.2** before starting this section.

- ___1. In the Integration Bus Toolkit, double-click on **Data Source Explorer**.



- ___2. The view will be expanded and you will see the databases. Right-click on **SAMPLE** and then click **Connect**.



If Connect is greyed out, then you are already connected. Move to step 5.

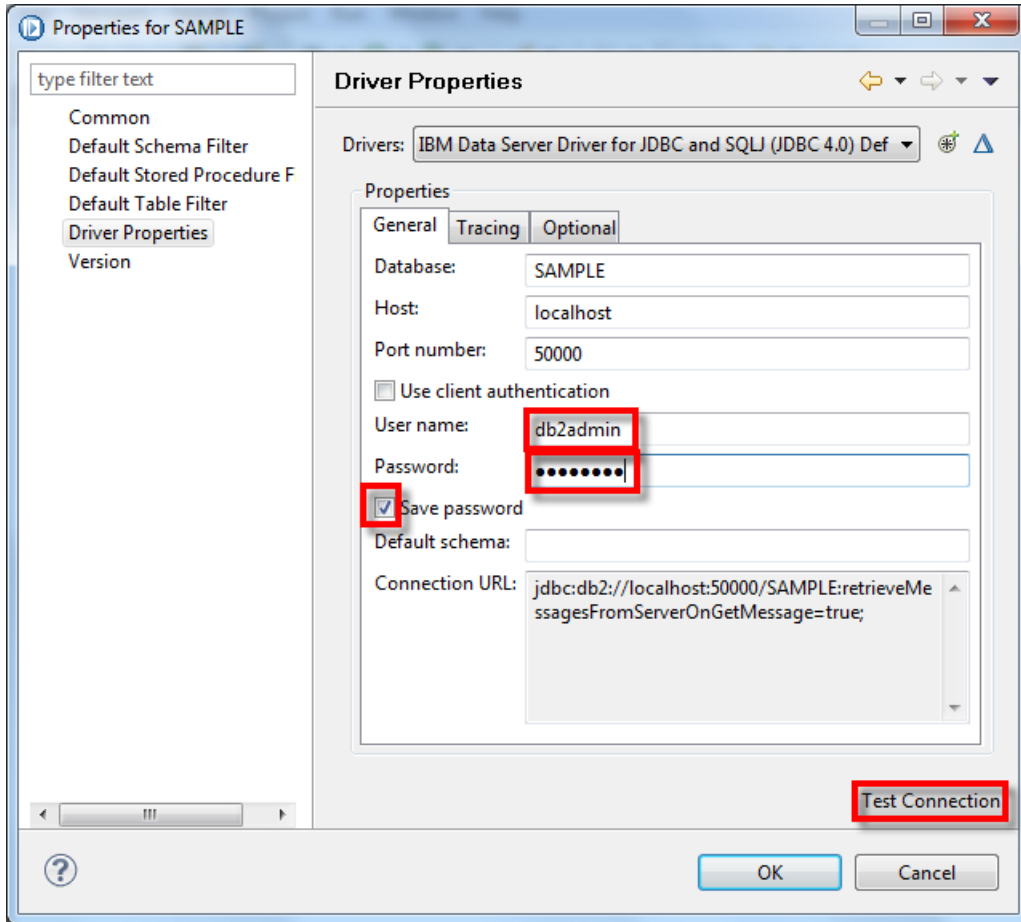
___3. To set up the connection to the database, keep the default fields and fill in the following fields:

User name: **db2admin**;

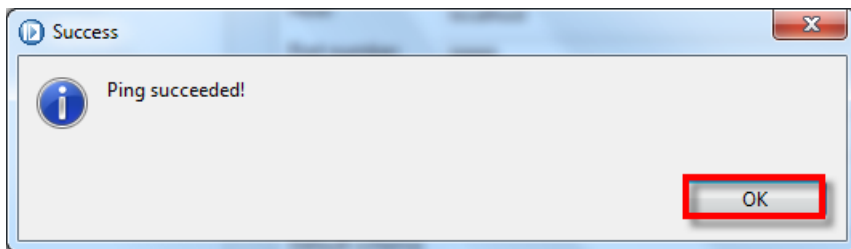
Password: **db8admin**;

Tick the box '**Save password**'

Click on '**Test connection**'.

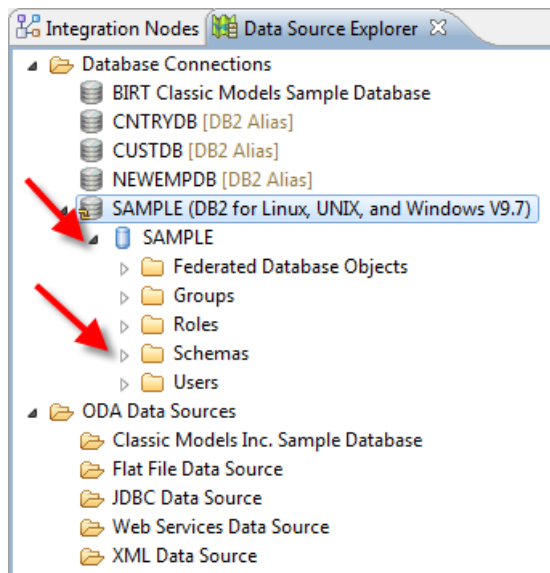


___4. If successful you will see the message as below. Click **OK**.

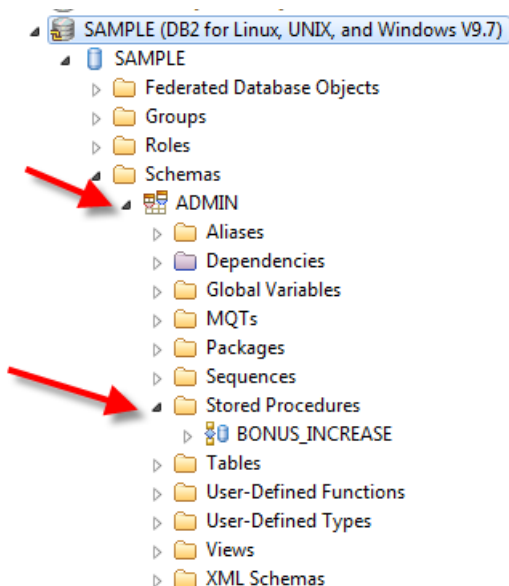


Click **OK** again to close the properties for SAMPLE.

__5. You are now connected to the **SAMPLE** database. Expand the **SAMPLE** database and then the **Schemas** folder.



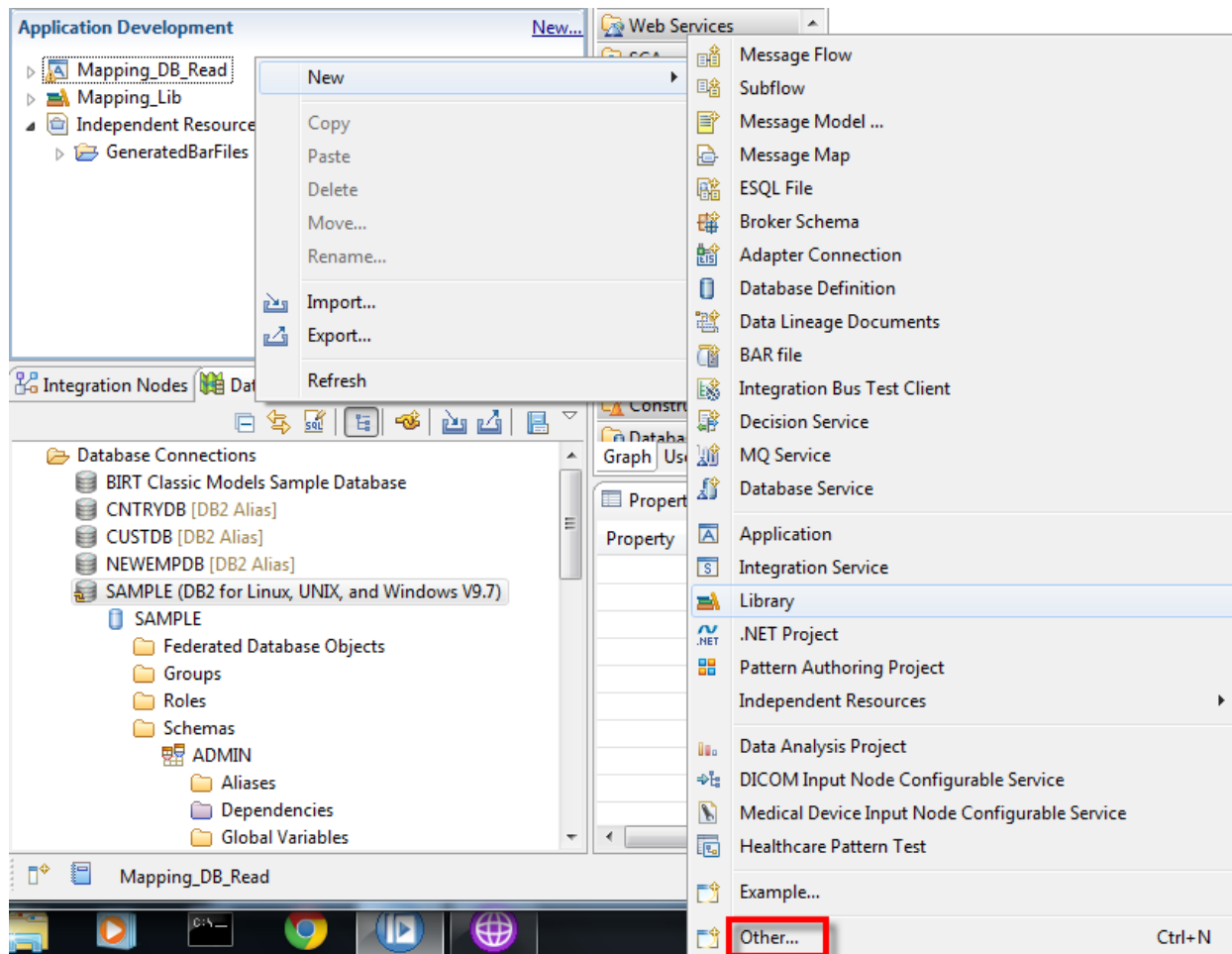
__6. Expand the schema **ADMIN** and then **Stored Procedures**. The database contains one sample Stored Procedure. This is where we will be adding our own procedure.



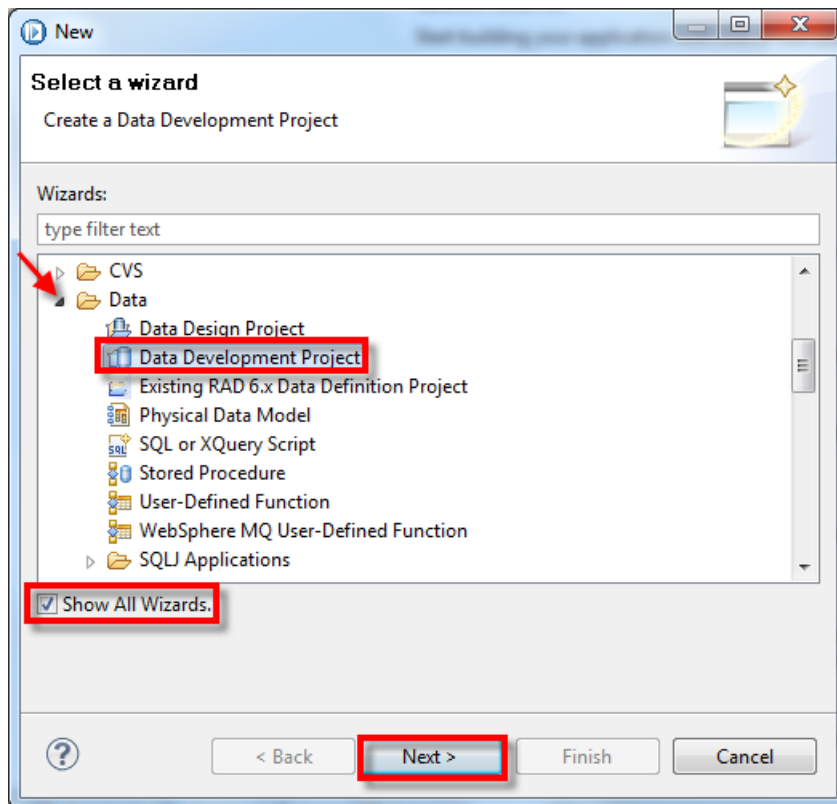
You will now create a new Stored Procedure that we will use in our application.

___7. Start by creating a new Data Development Project. If you still have the Data Source Explorer expanded, **double-click** on its tab to return to the default view of the Integration development perspective.

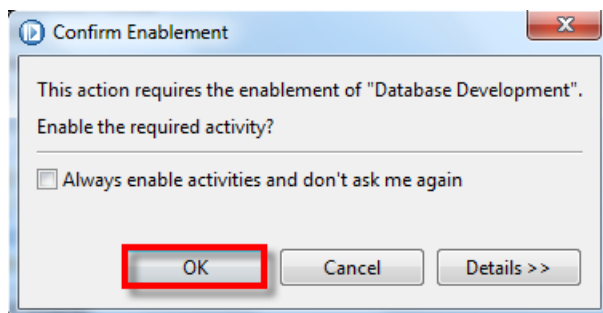
In **Application Development**, right-click in a blank area, then select **New → Other**.



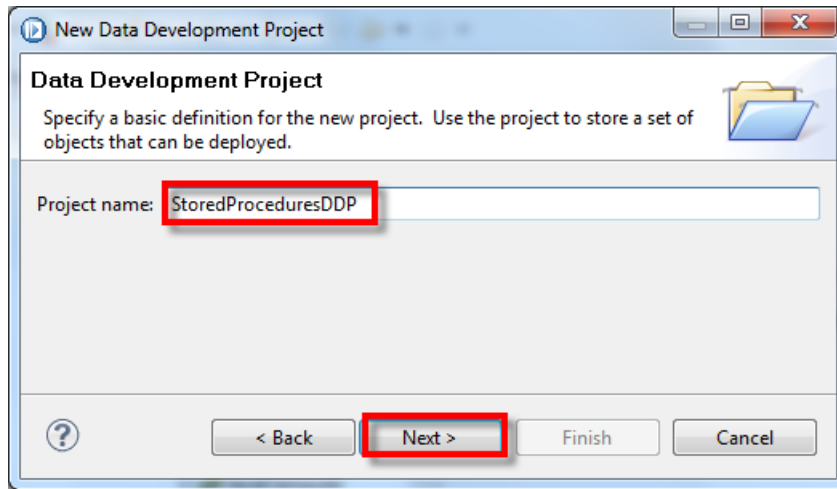
__8. In the opened window, expand **Data** and select **Data Development Project**, then click **Next** (check **Show All Wizards** if not seen).



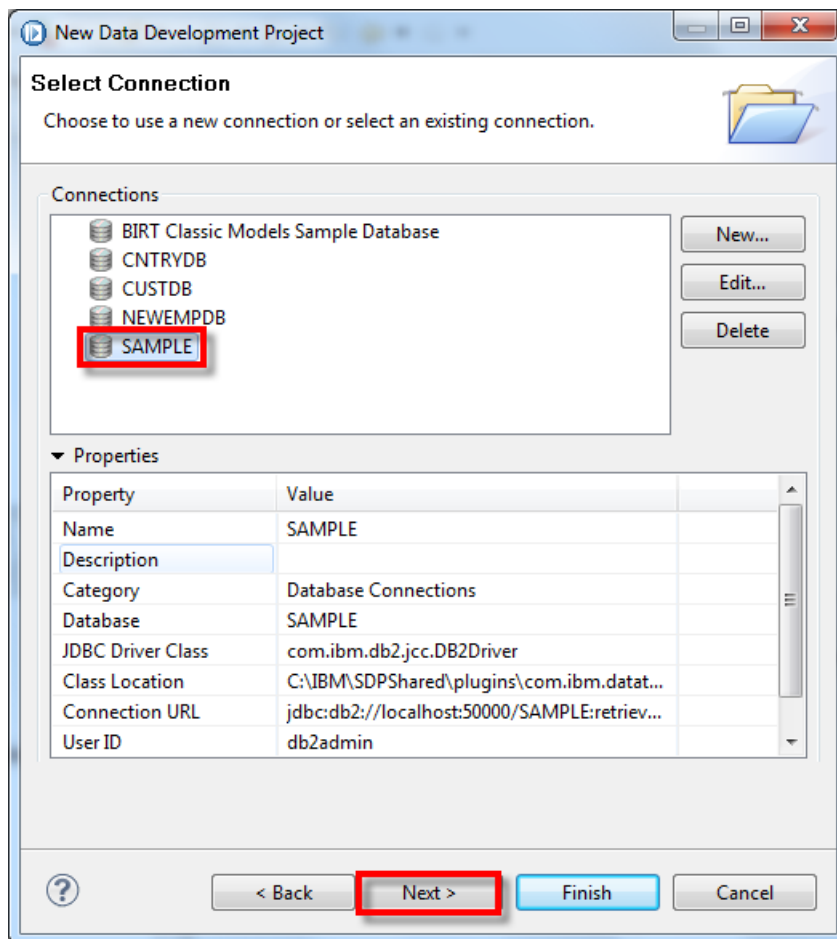
__9. If you see the following message, click **OK**.



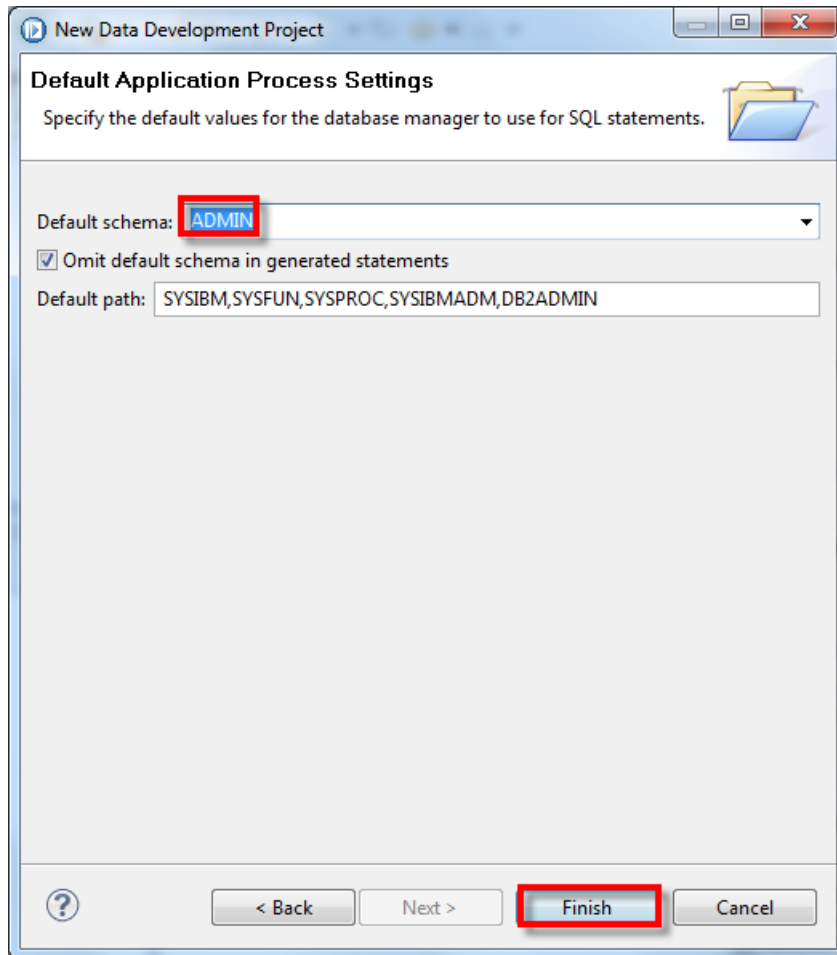
__10. Name the new Data Development Project **StoredProceduresDDP** and click **Next**.



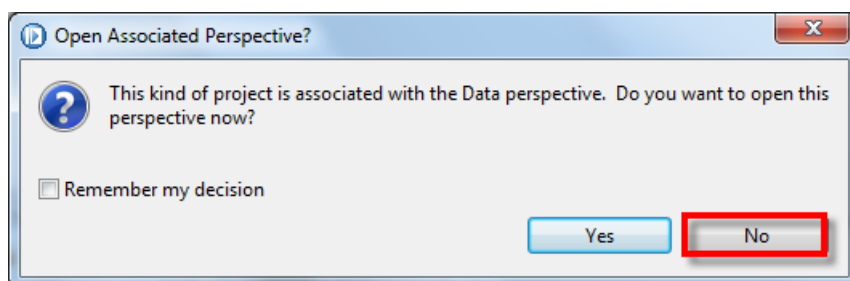
__11. On the following screen, select the previously created **SAMPLE** connection. Click on **Next**.



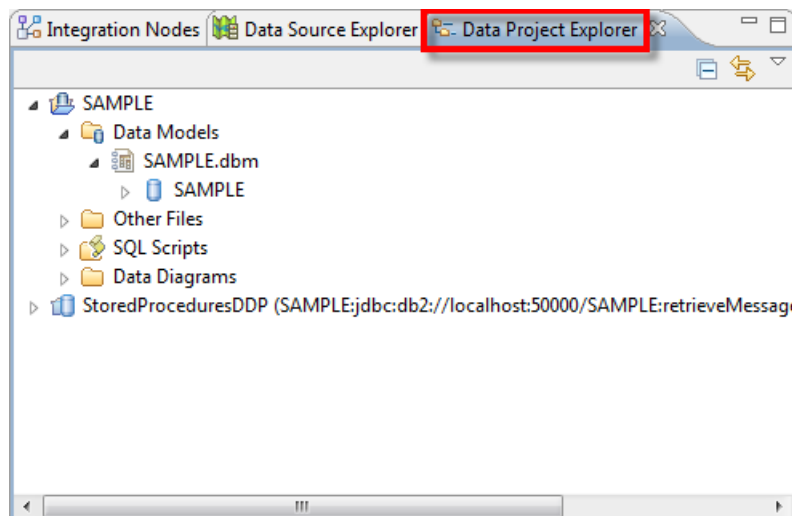
__12. Enter **ADMIN** as the **Default Schema**, and then click on **Finish** to create the Project.



__13. On the opened dialog, click **No** to stay in the Integration Development perspective.

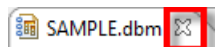


__14. The **Data Project Explorer** view will be added to the Application Development perspective (bottom left – next to Data Source Explorer).

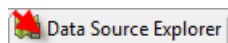


If you do not see the view, go to **Window → Show View** and click on **Data Project Explorer**.

__15. If the SAMPLE.dbm file is opened, close the **SAMPLE.dbm** file.

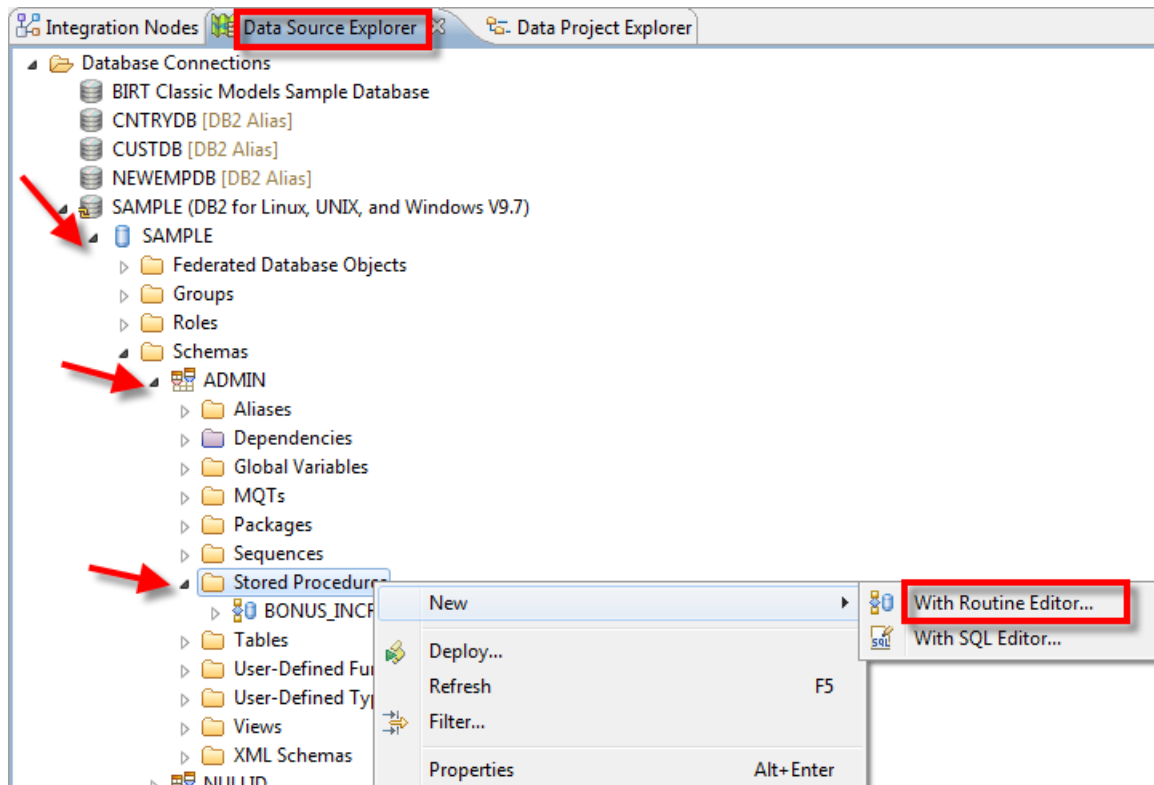


__16. Double-click on **Data Source Explorer** to view in full screen.



__17. You should still be connected to the SAMPLE database. If you have been disconnected, right-click on **SAMPLE** and then **Connect**. Since you created a database connection earlier, you do not have to go through the set up step.

__18. Expand **SAMPLE** → **Schemas** → **ADMIN**. To open the editor for creating a new procedure, right-click on **Stored Procedures** then **New**→**With Routine Editor**....



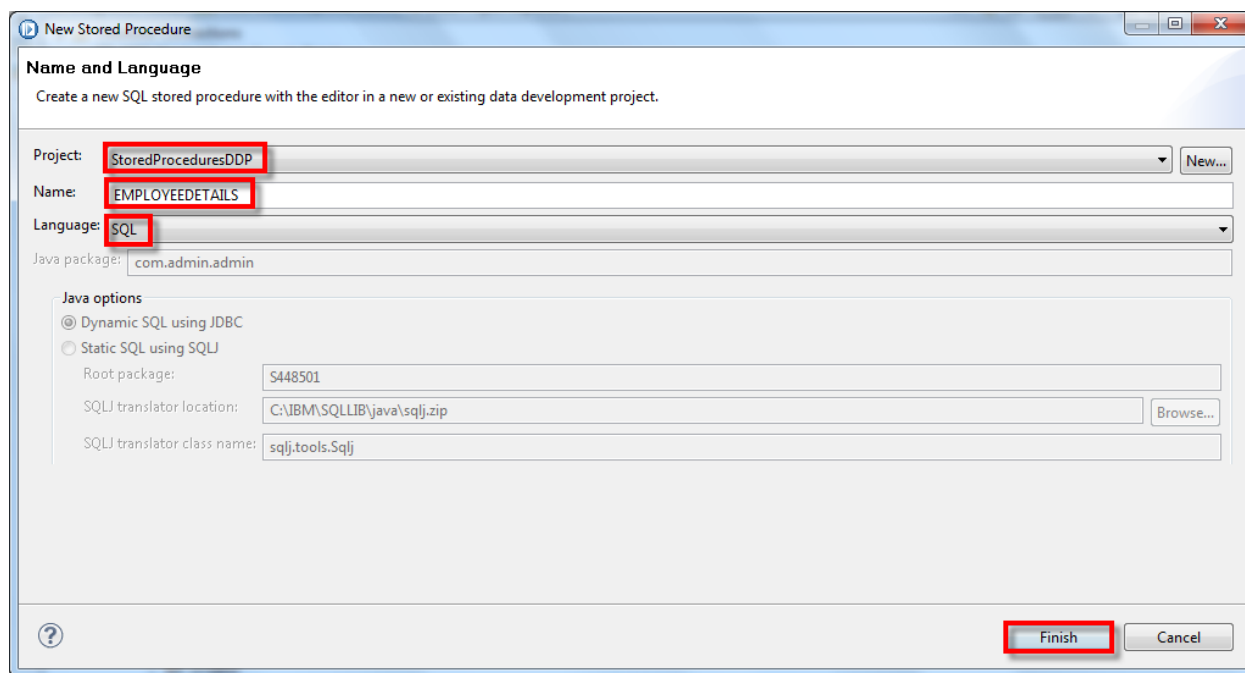
__19. In the newly opened window, you should edit the following fields:

Project – **StoredProceduresDDP** (select from drop-down if not automatically filled in)

Name – **EMPLOYEEDETAILS**

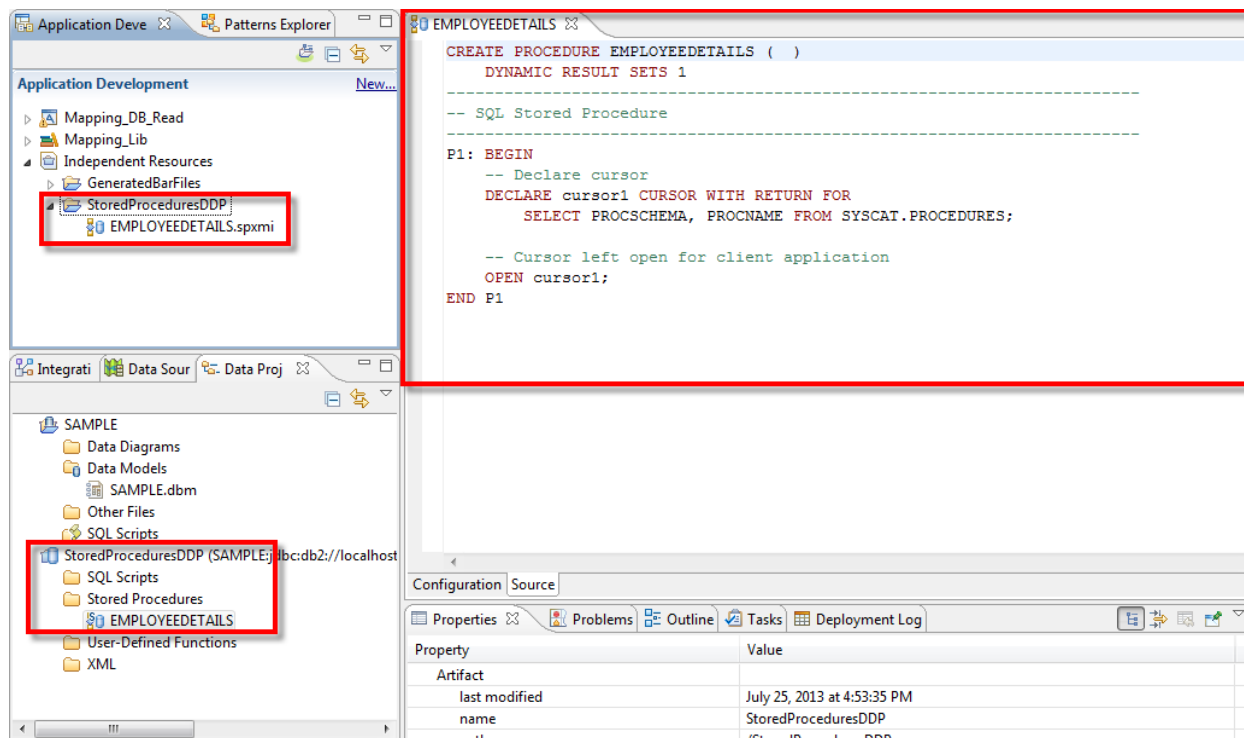
Language – **SQL** (select from drop-down)

Once finished you should have a similar view as below:



20. Click **Finish**.

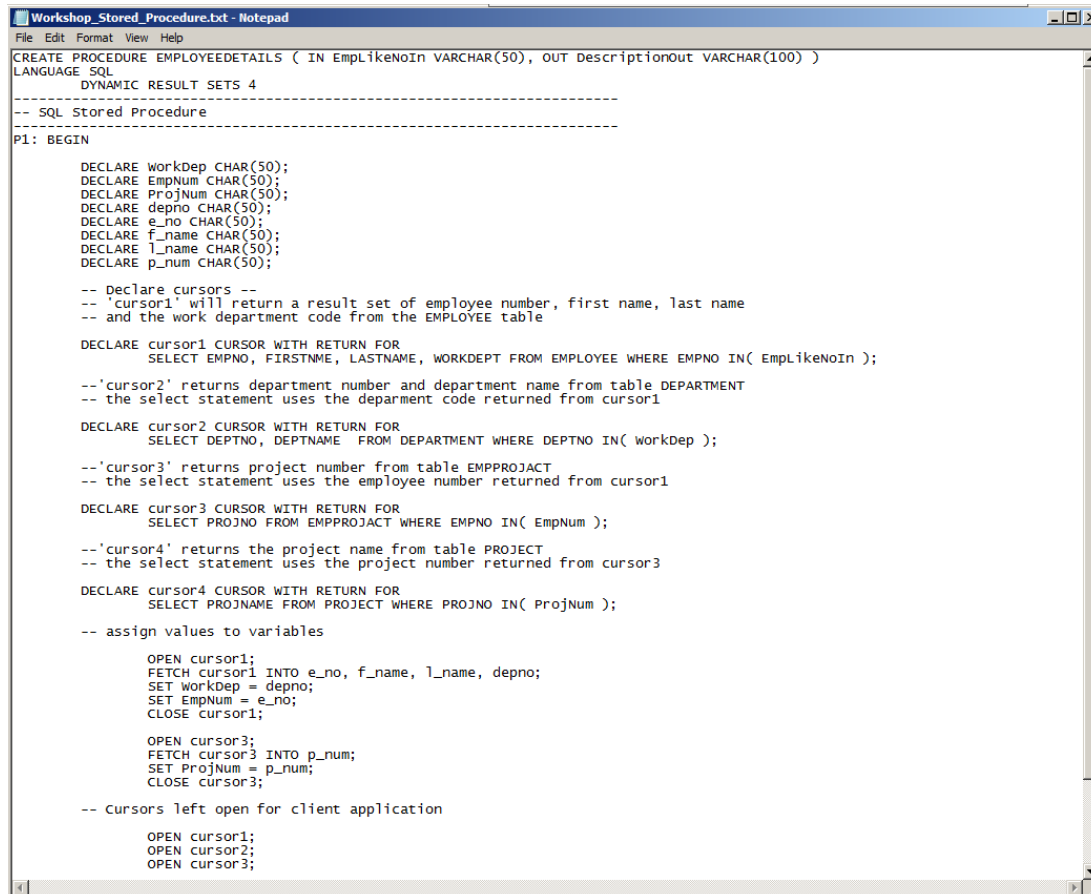
21. Integration Bus Toolkit creates the procedure and opens the editor with a few lines already written. You will also notice that the Toolkit has created a new file in the Application Development view – **EMPLOYEEDETAILS.spxmi**.



22. The procedure that we will use in this lab has been written for you.

Using Windows Explorer, navigate to

C:\student\Mapping_database\StoredProcedures\Procedure and open the text file **Workshop_Stored_Procedure.txt**



```

CREATE PROCEDURE EMPLOYEEDETAILS ( IN EmplikeNoIn VARCHAR(50), OUT DescriptionOut VARCHAR(100) )
LANGUAGE SQL
    DYNAMIC RESULT SETS 4
-----
-- SQL Stored Procedure
-----
P1: BEGIN

    DECLARE workDep CHAR(50);
    DECLARE EmpNum CHAR(50);
    DECLARE ProjNum CHAR(50);
    DECLARE depno CHAR(50);
    DECLARE e_no CHAR(50);
    DECLARE f_name CHAR(50);
    DECLARE l_name CHAR(50);
    DECLARE p_num CHAR(50);

    -- Declare cursors --
    -- 'cursor1' will return a result set of employee number, first name, last name
    -- and the work department code from the EMPLOYEE table

    DECLARE cursor1 CURSOR WITH RETURN FOR
        SELECT EMPNO, FIRSTNME, LASTNAME, WORKDEPT FROM EMPLOYEE WHERE EMPNO IN( EmplikeNoIn );

    -- 'cursor2' returns department number and department name from table DEPARTMENT
    -- the select statement uses the department code returned from cursor1

    DECLARE cursor2 CURSOR WITH RETURN FOR
        SELECT DEPTNO, DEPTNAME FROM DEPARTMENT WHERE DEPTNO IN( workDep );

    -- 'cursor3' returns project number from table EMPPROJECT
    -- the select statement uses the employee number returned from cursor1

    DECLARE cursor3 CURSOR WITH RETURN FOR
        SELECT PROJNO FROM EMPPROJECT WHERE EMPNO IN( EmpNum );

    -- 'cursor4' returns the project name from table PROJECT
    -- the select statement uses the project number returned from cursor3

    DECLARE cursor4 CURSOR WITH RETURN FOR
        SELECT PROJNAME FROM PROJECT WHERE PROJNO IN( ProjNum );

    -- assign values to variables

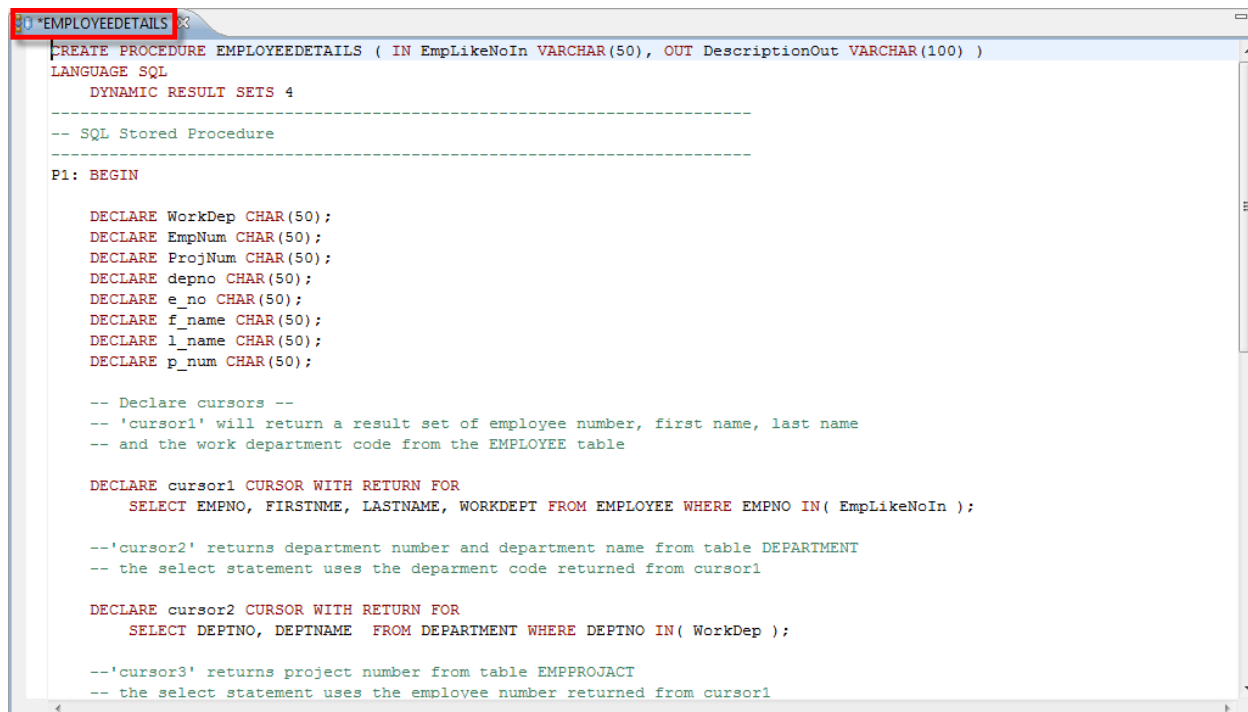
    OPEN cursor1;
    FETCH cursor1 INTO e_no, f_name, l_name, depno;
    SET workDep = depno;
    SET EmpNum = e_no;
    CLOSE cursor1;

    OPEN cursor3;
    FETCH cursor3 INTO p_num;
    SET ProjNum = p_num;
    CLOSE cursor3;

    -- Cursors left open for client application

    OPEN cursor1;
    OPEN cursor2;
    OPEN cursor3;
  
```

__23. Copy the entire text (**CTRL-A / CTRL – C**) and paste it (**CTRL-V**) back in the Integration Bus Toolkit editor (delete the previously generated default code). **Double-click** on the tab with the Procedure name to expand the window.



```
CREATE PROCEDURE EMPLOYEEDETAILS ( IN EmpLikeNoIn VARCHAR(50), OUT DescriptionOut VARCHAR(100) )
LANGUAGE SQL
DYNAMIC RESULT SETS 4
-----
-- SQL Stored Procedure
-----
P1: BEGIN

  DECLARE WorkDep CHAR(50);
  DECLARE EmpNum CHAR(50);
  DECLARE ProjNum CHAR(50);
  DECLARE depno CHAR(50);
  DECLARE e_no CHAR(50);
  DECLARE f_name CHAR(50);
  DECLARE l_name CHAR(50);
  DECLARE p_num CHAR(50);

  -- Declare cursors --
  -- 'cursor1' will return a result set of employee number, first name, last name
  -- and the work department code from the EMPLOYEE table

  DECLARE cursor1 CURSOR WITH RETURN FOR
    SELECT EMPNO, FIRSTNAME, LASTNAME, WORKDEPT FROM EMPLOYEE WHERE EMPNO IN( EmpLikeNoIn );

  --'cursor2' returns department number and department name from table DEPARTMENT
  -- the select statement uses the department code returned from cursor1

  DECLARE cursor2 CURSOR WITH RETURN FOR
    SELECT DEPTNO, DEPTNAME FROM DEPARTMENT WHERE DEPTNO IN( WorkDep );

  --'cursor3' returns project number from table EMPPROJACT
  -- the select statement uses the employee number returned from cursor1
```

__24. Close the Notepad window.

__25. The Stored Procedure is written in SQL. It starts by defining the input and output parameters. Then it defines the number of “result sets” which are the values returned from the “Select” statements that follow in the procedure.

The **cursors** are dynamic variables that store the return results before being sent to the requesting application.

Please take a moment to go through the procedure and the comments for each operation.

In summary, the procedure takes an input value for “Employee Number” and returns data for the selected employee from four different tables in the SAMPLE database. Within the procedure selected values from one table are used for selecting rows from other tables.

This shows how you can make one call to the database and have a number of operations completed on the database server before the result is returned back.

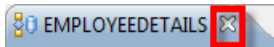
The first “select” statement returns three columns from the EMPLOYEE table.

When finished reviewing, **double-click** on the tab **EMPLOYEEDETAILS** to return to Integration Development perspective.

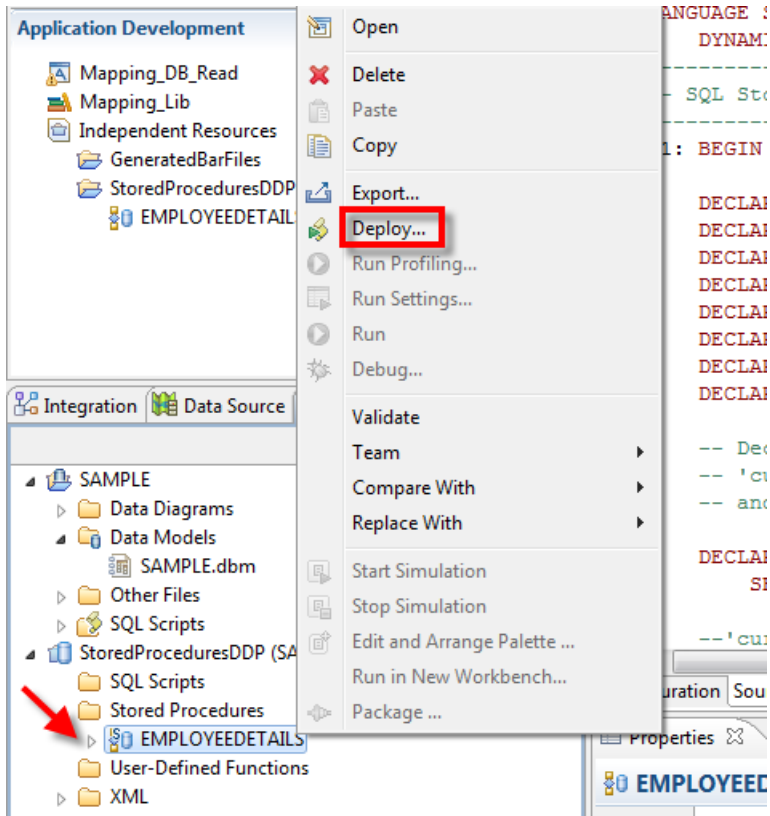
Make sure you save the new procedure with **Ctrl+S**.



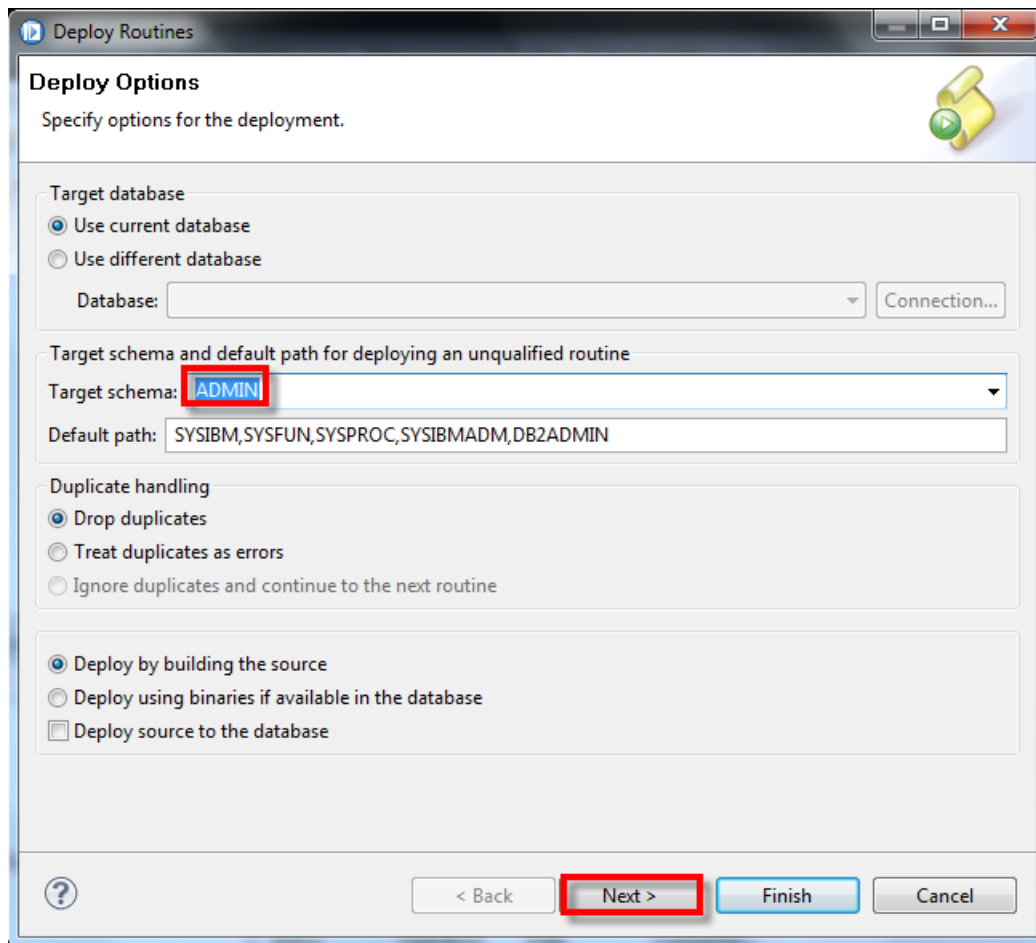
__26. Close the **EMPLOYEEDETAILS** Stored Procedure.



__27. In the **Data Project Explorer**, right-click on the newly created procedure **EMPLOYEEDETAILS** and then **Deploy...**



28. You are deploying the procedure to the SAMPLE database and a “Deploy Options” window will open. Ensure **ADMIN** is selected as the **Target schema**, and then click **Next**.

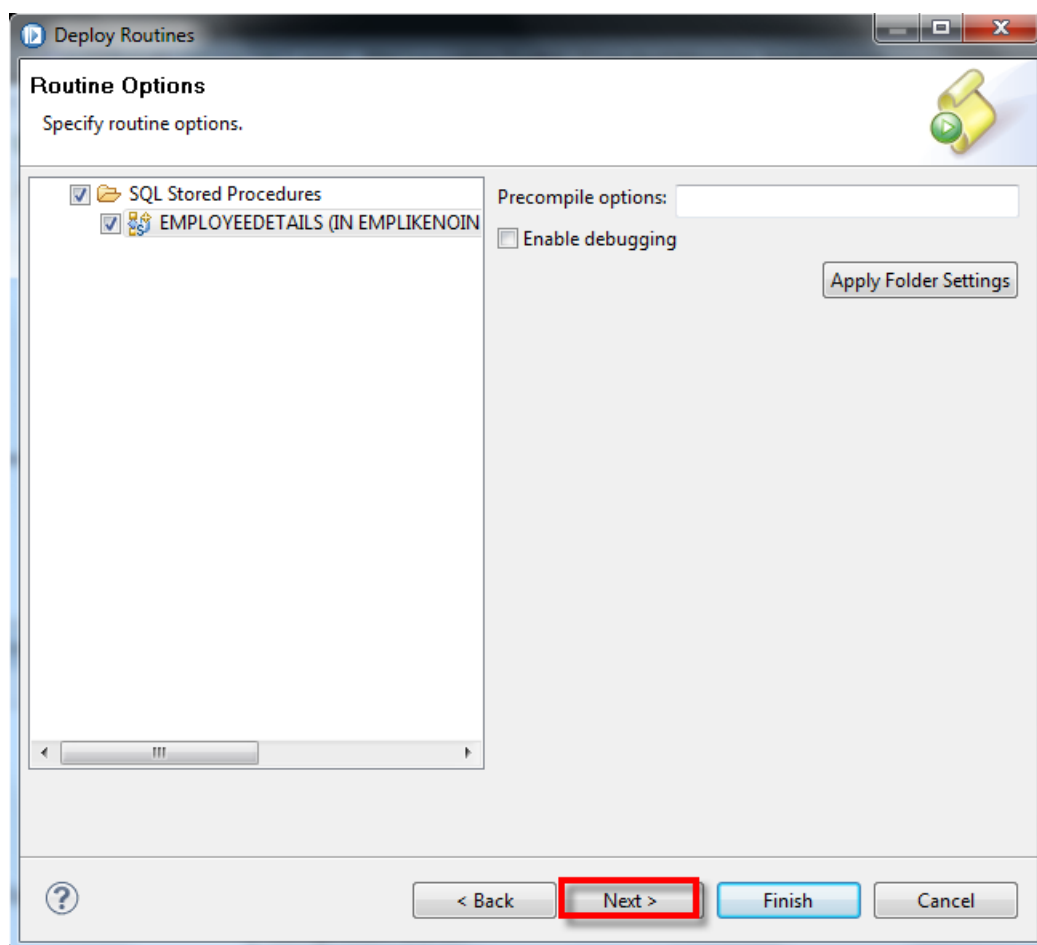


The image shows a Windows-style dialog box titled "Deploy Routines" with a subtitle "Deploy Options". The main instruction is "Specify options for the deployment." The dialog is divided into several sections:

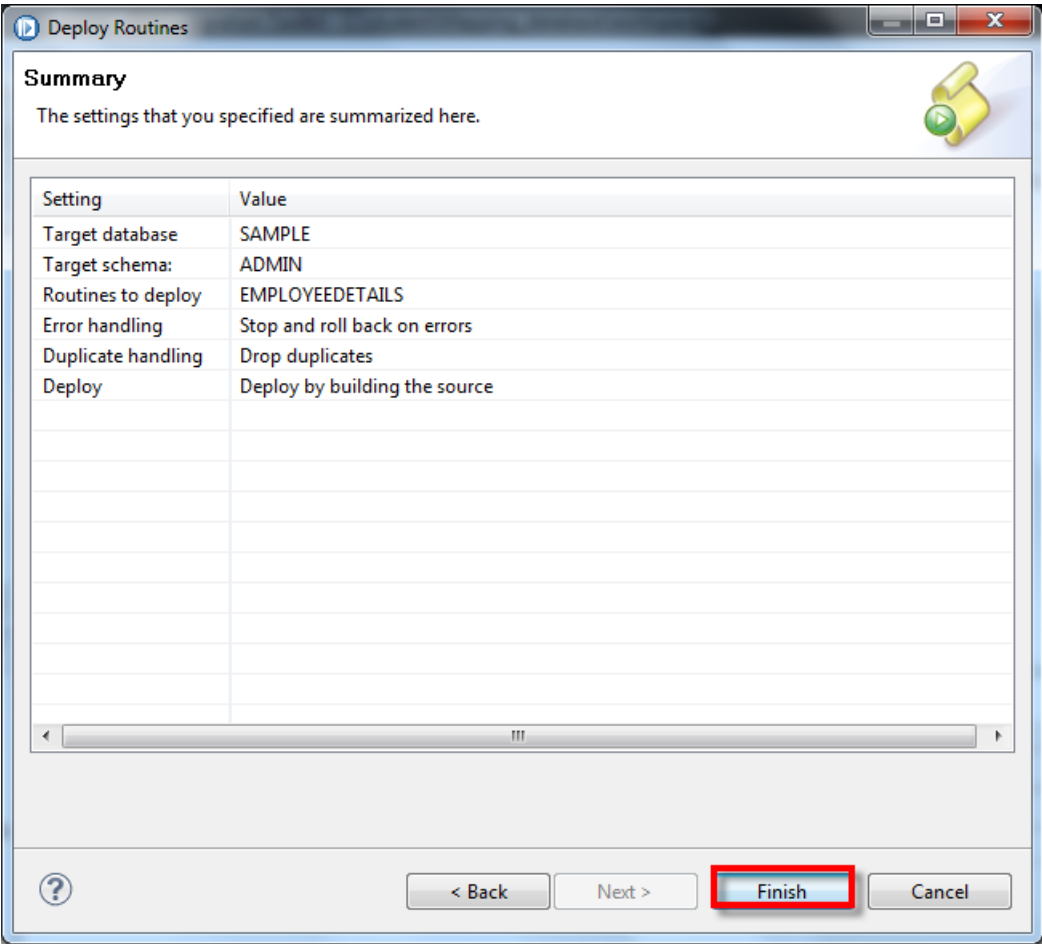
- Target database:** Contains two radio buttons: "Use current database" (selected) and "Use different database". Below is a "Database:" text box and a "Connection..." button.
- Target schema and default path for deploying an unqualified routine:** Contains a "Target schema:" dropdown menu with "ADMIN" selected (highlighted with a red box) and a "Default path:" text box containing "SYSIBM,SYSFUN,SYSPROC,SYSIBMADM,DB2ADMIN".
- Duplicate handling:** Contains three radio buttons: "Drop duplicates" (selected), "Treat duplicates as errors", and "Ignore duplicates and continue to the next routine".
- Deployment options:** Contains three checkboxes: "Deploy by building the source" (selected), "Deploy using binaries if available in the database", and "Deploy source to the database".

At the bottom, there is a help icon (?), a "< Back" button, a "Next >" button (highlighted with a red box), a "Finish" button, and a "Cancel" button.

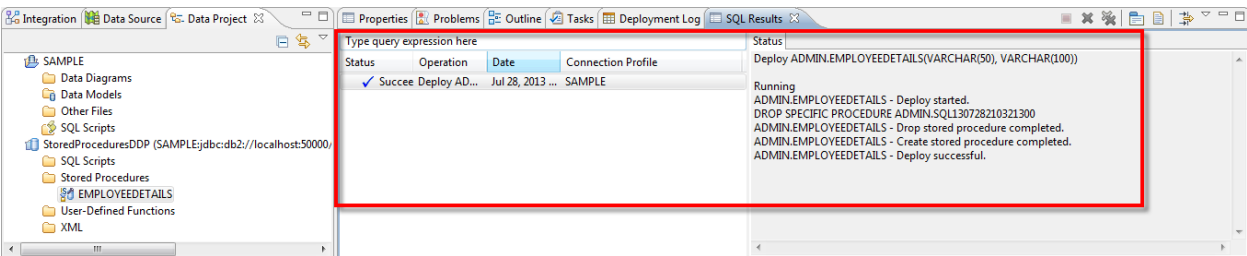
29. Click **Next** again.



__30. Finally, click **Finish** on the Summary window.



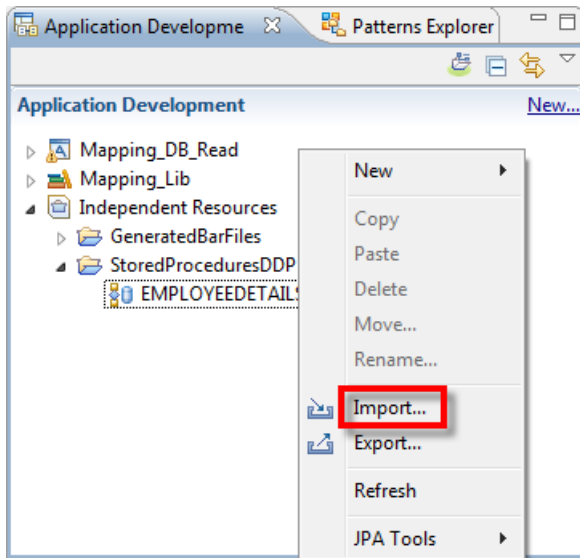
__31. In the bottom right of the perspective, a “**SQL Results**” tab will open showing a successful deployment of the procedure.



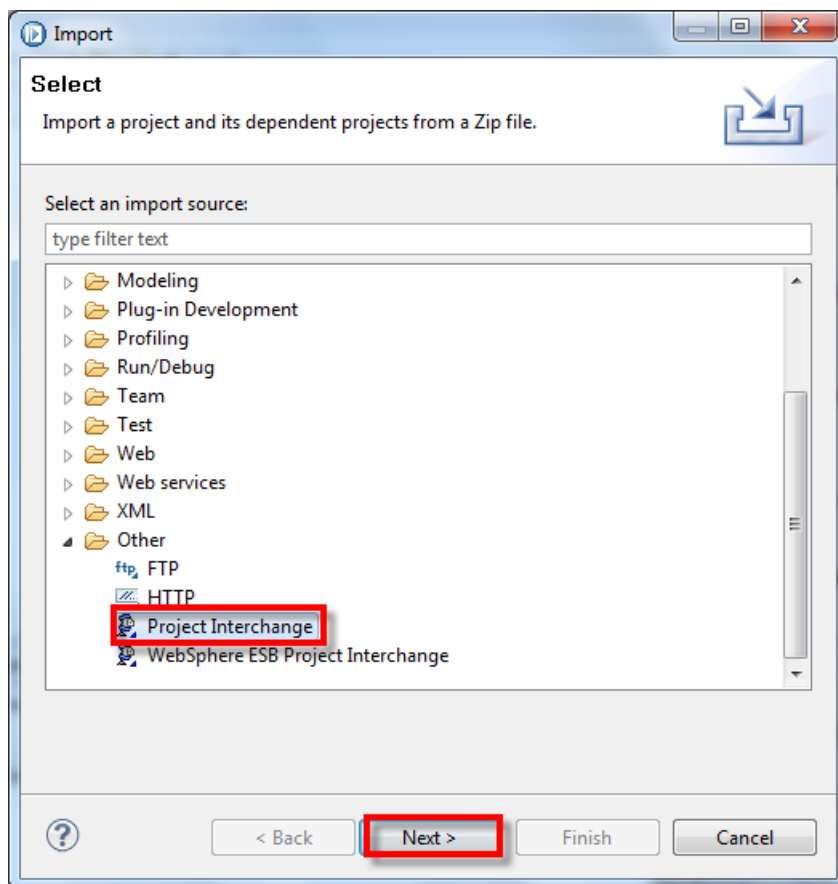
3.4.2 Stored Procedures application

You will now import a simple application and will complete all the necessary mappings.

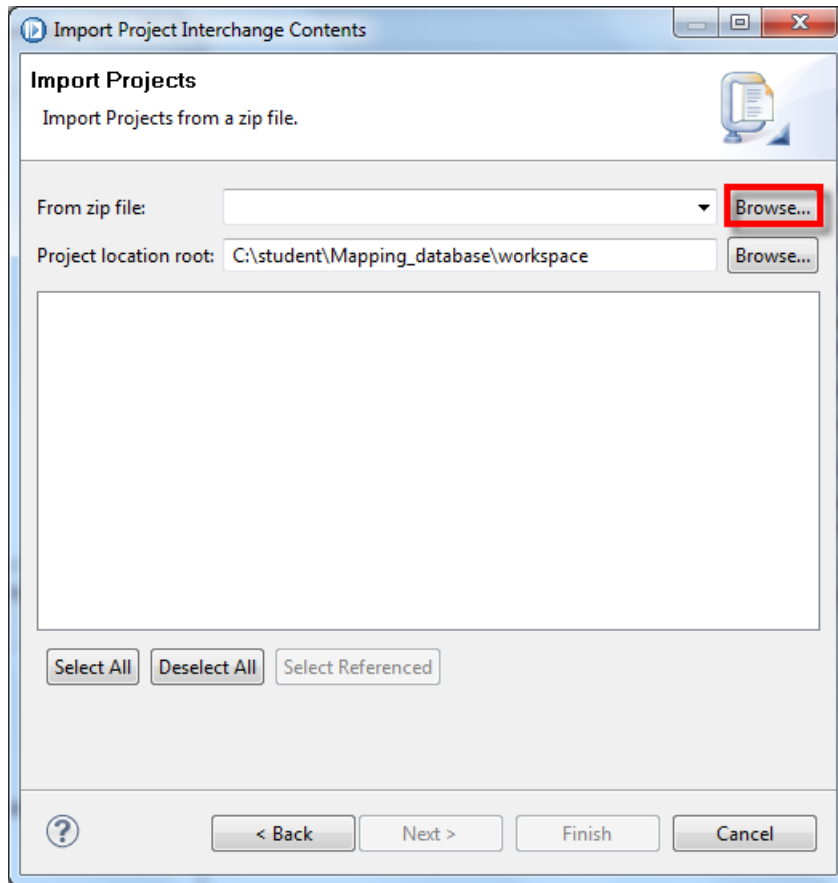
- ___1. In the Application Development view, **right-click** in an empty area and select **Import....**



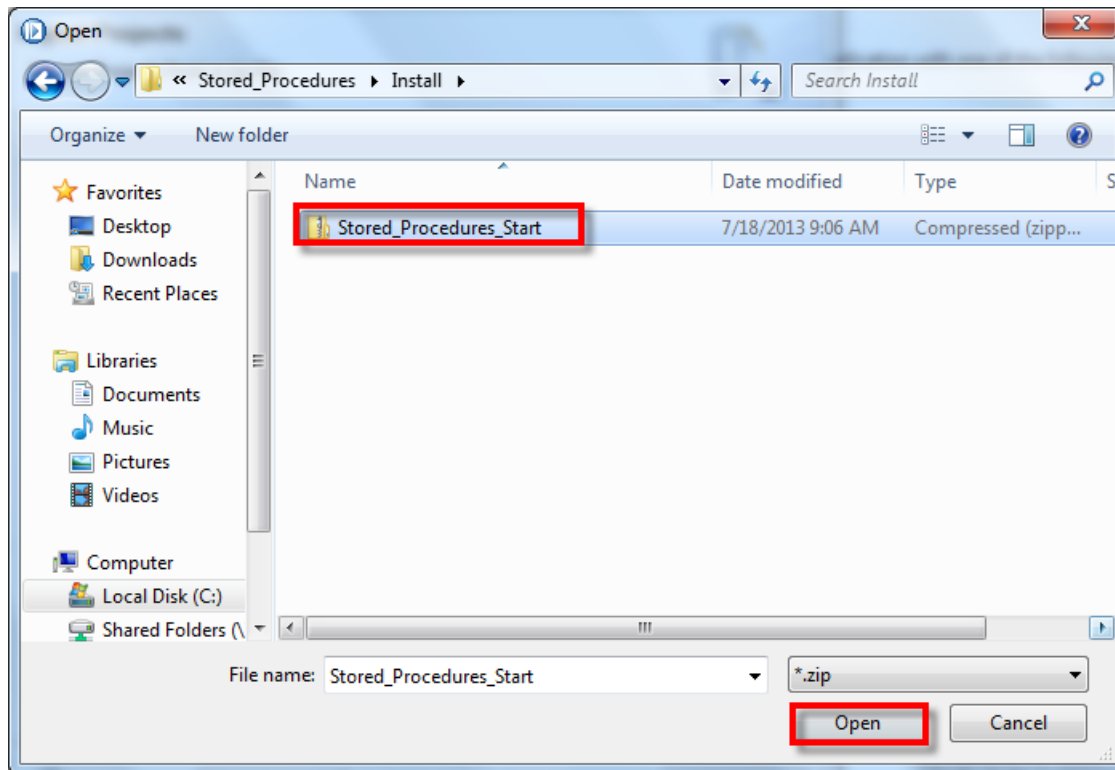
- ___2. In the Import window, expand **Other** and select **Project interchange**. Click **Next**.



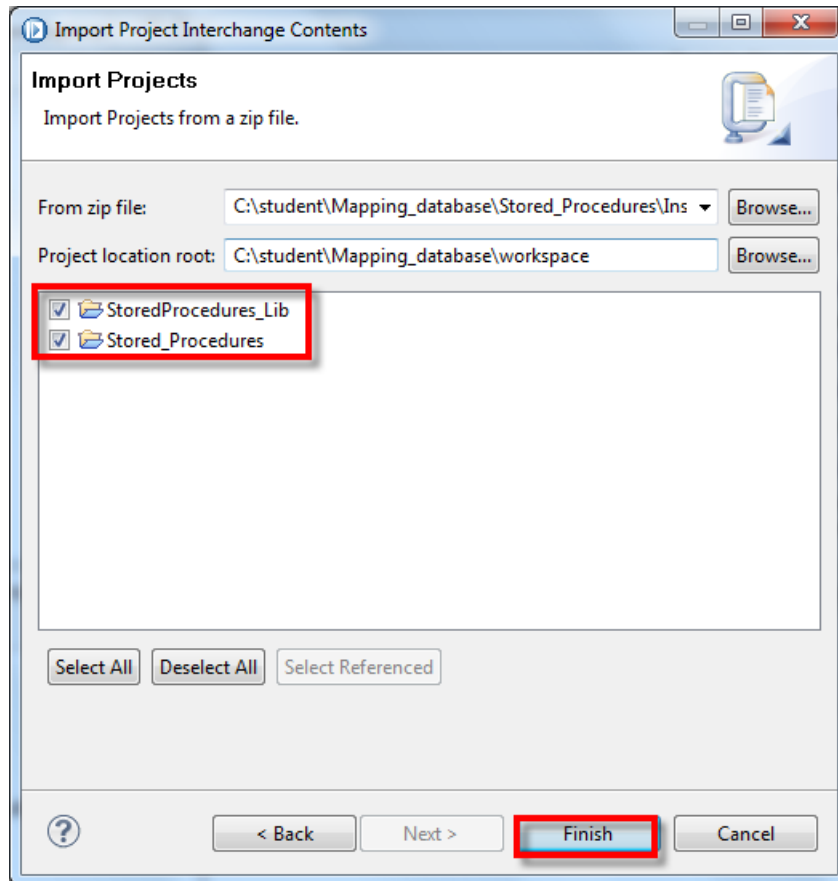
___3. On the next screen, click **Browse...** for **From zip file**.



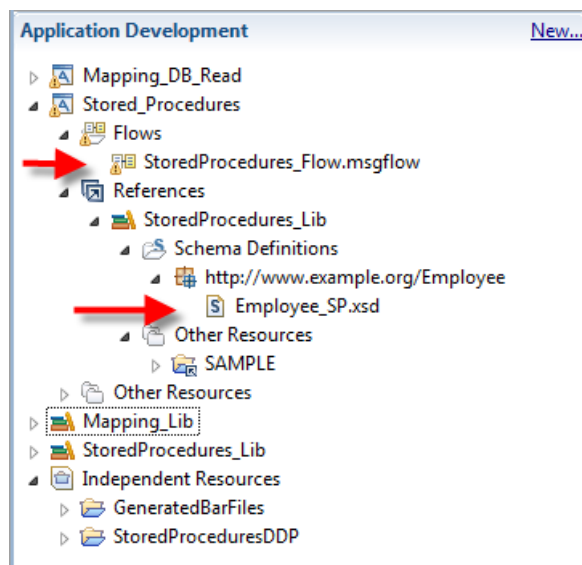
4. Navigate to **C:\student\Mapping_database\Stored_Procedures\Install** and select **Stored_Procedures_Start.zip**. Click **Open**.



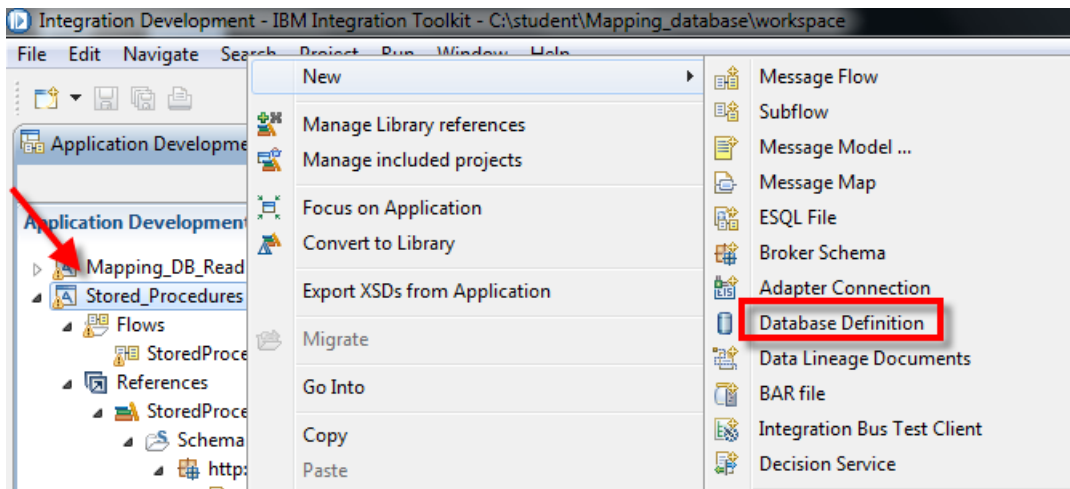
__5. Ensure both folders are selected. Click **Finish** to complete the import:



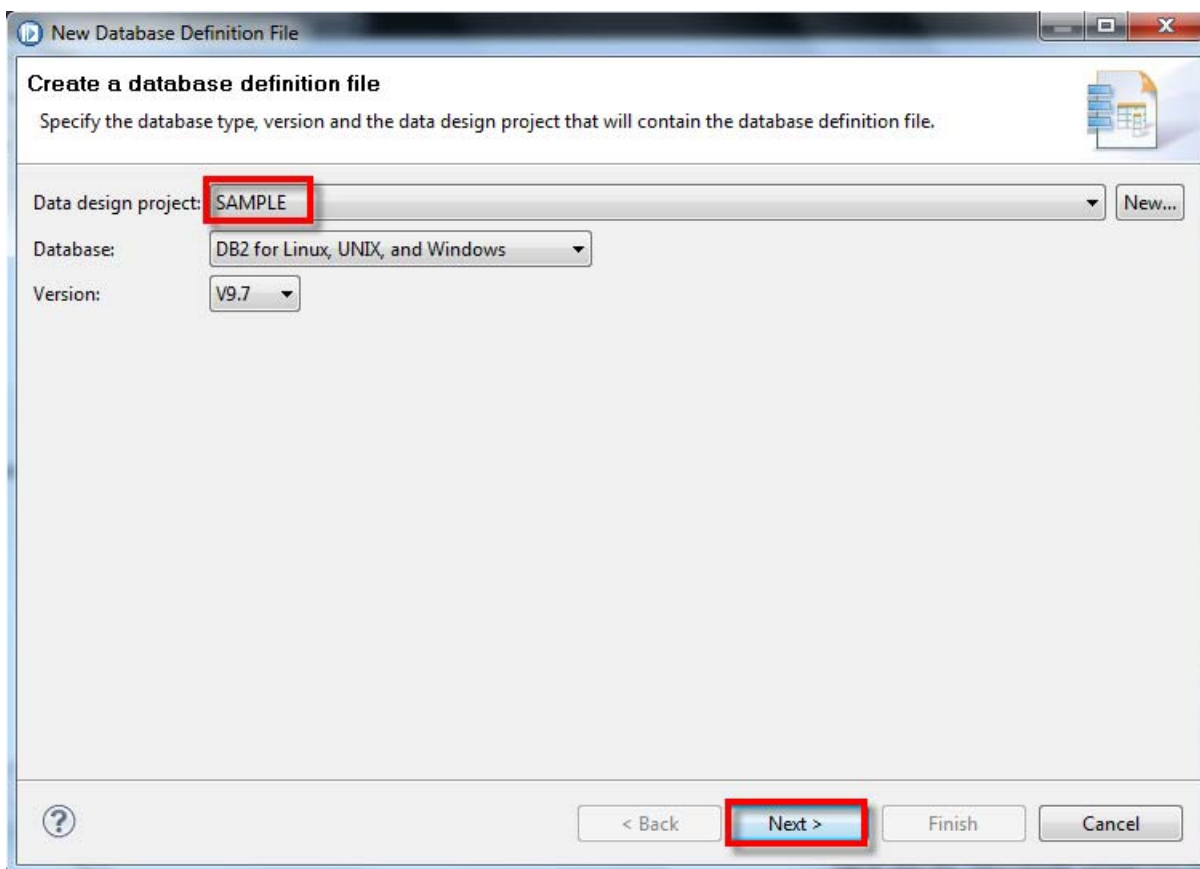
__6. Review the imported application. You will see the application message flow and its library which includes the schema file that is being used.



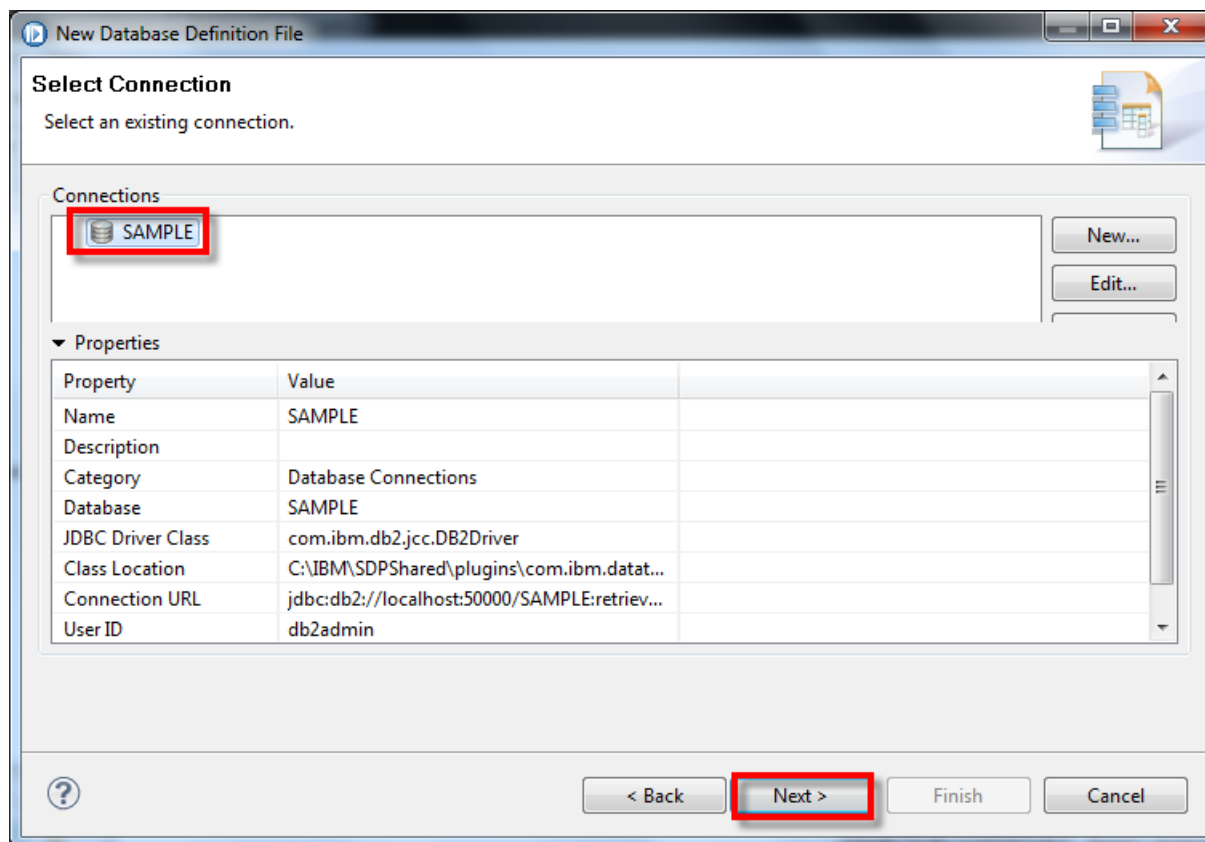
__7. You will now create a new database definition that the application will use. **Right-click** on the imported application and select **New→Database definition**.



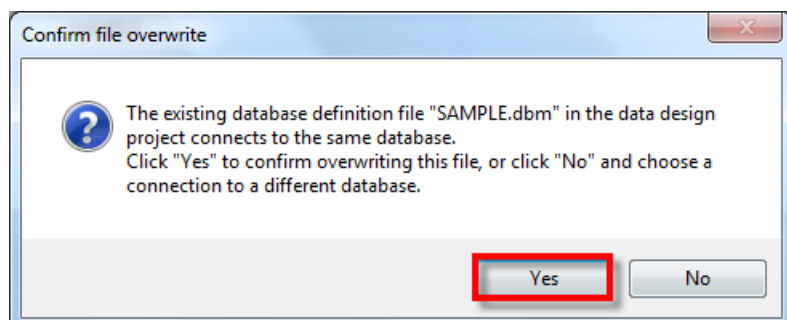
__8. Use the drop-down list to select the **SAMPLE** Data design project created earlier. Make sure DB2 and V9.7 are selected. Click **Next**.



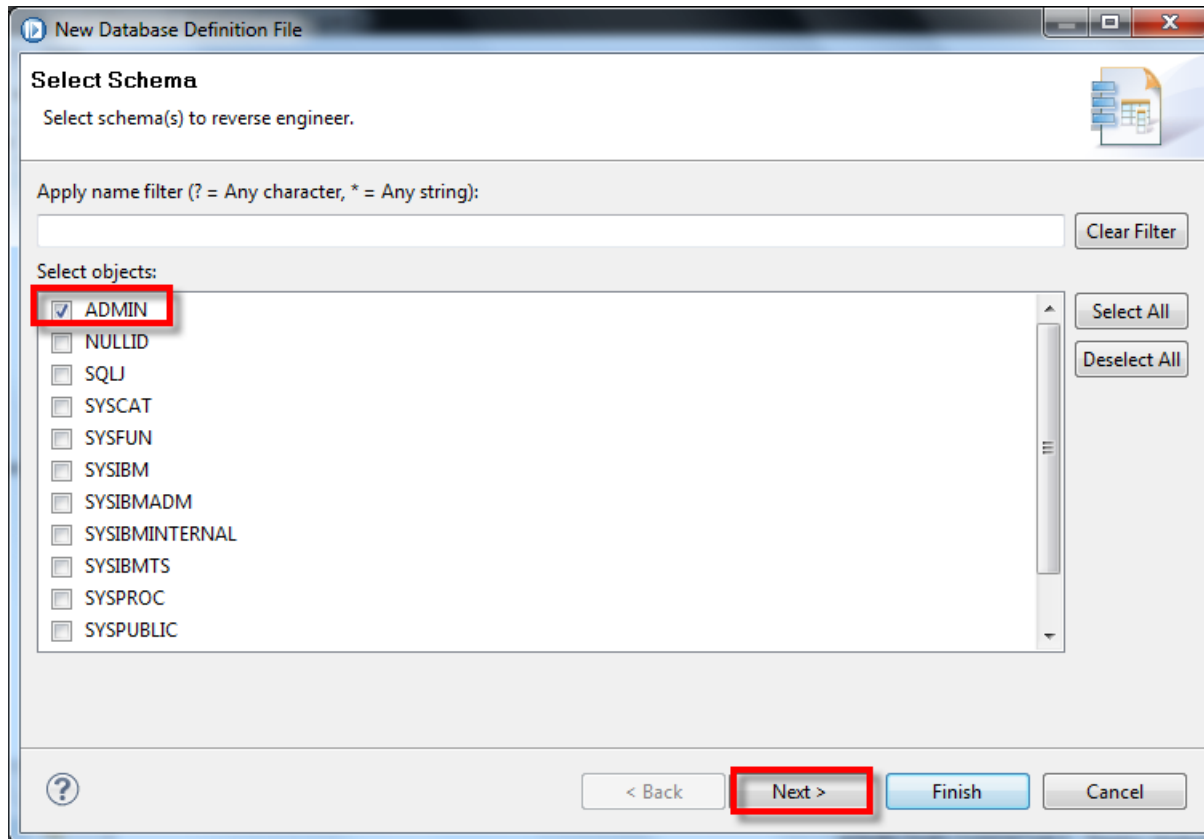
__9. On the next screen select **SAMPLE** (we created it earlier to connect to the database) and then **Next**.



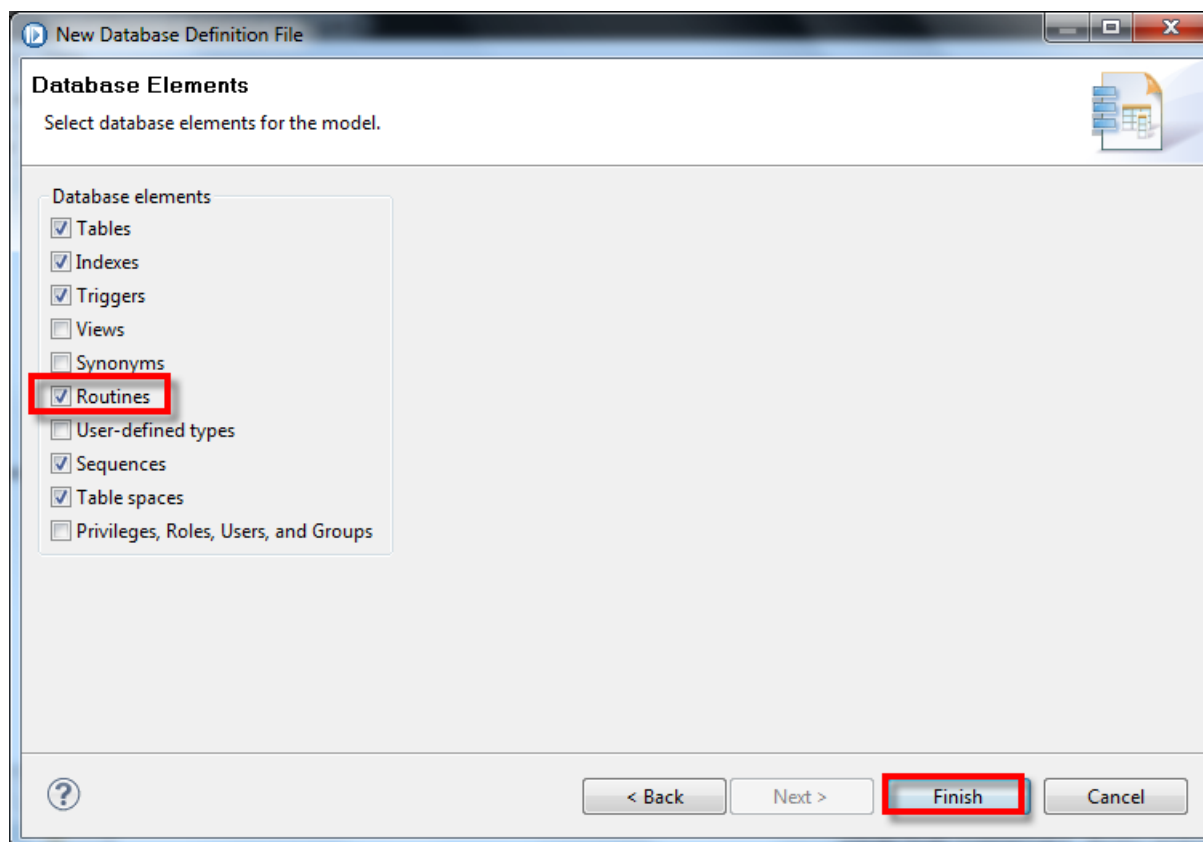
__10. Select **Yes** to confirm the updating of the existing file.



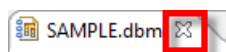
__11. Select the **ADMIN** schema and click **Next**.



__12. For **Database Elements**, make sure that you check **Routines** and then click **Finish**.

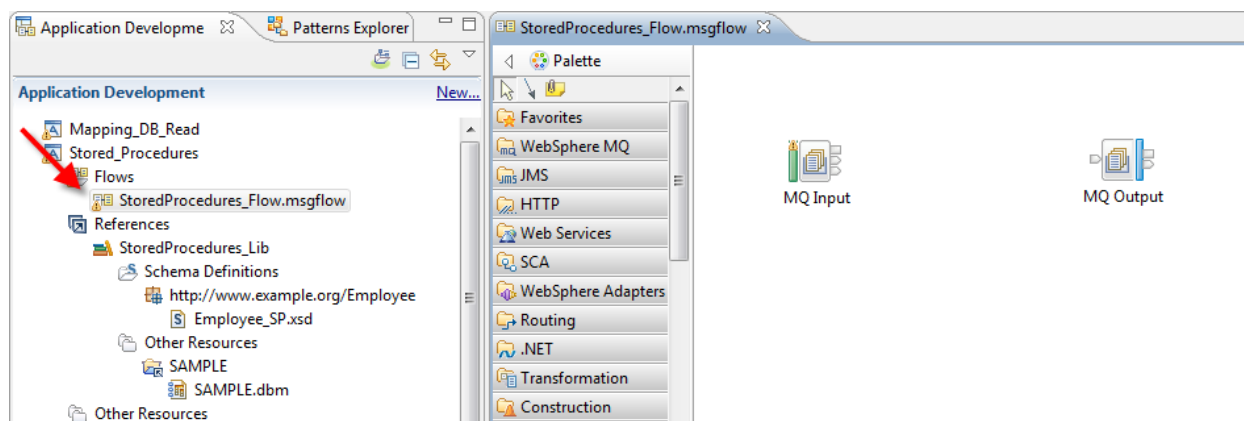


__13. Close the file **SAMPLE.dbm** file.

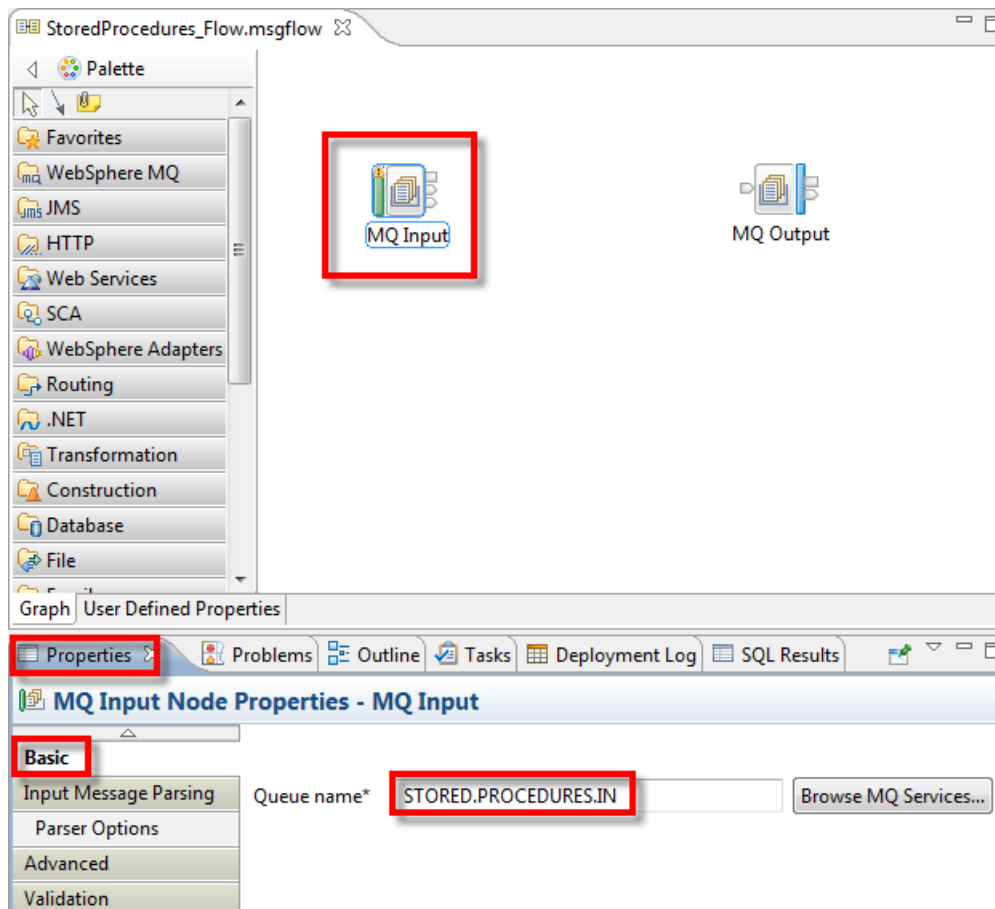


Important: Please note that if you create/edit/ update a Stored Procedure you will have to go through steps 7 – 13 again, in order to be able to work with the most current procedures. There is no “Refresh” capability.

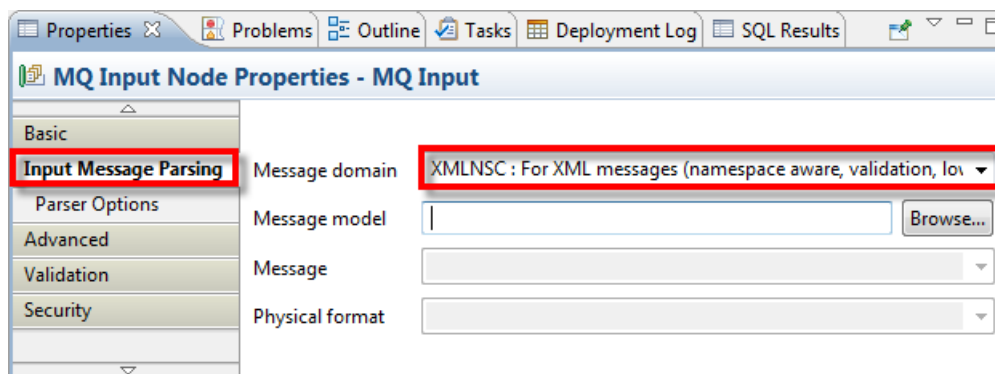
__14. Double-click on **StoredProcedures_Flow.msgflow** to open the application message flow.



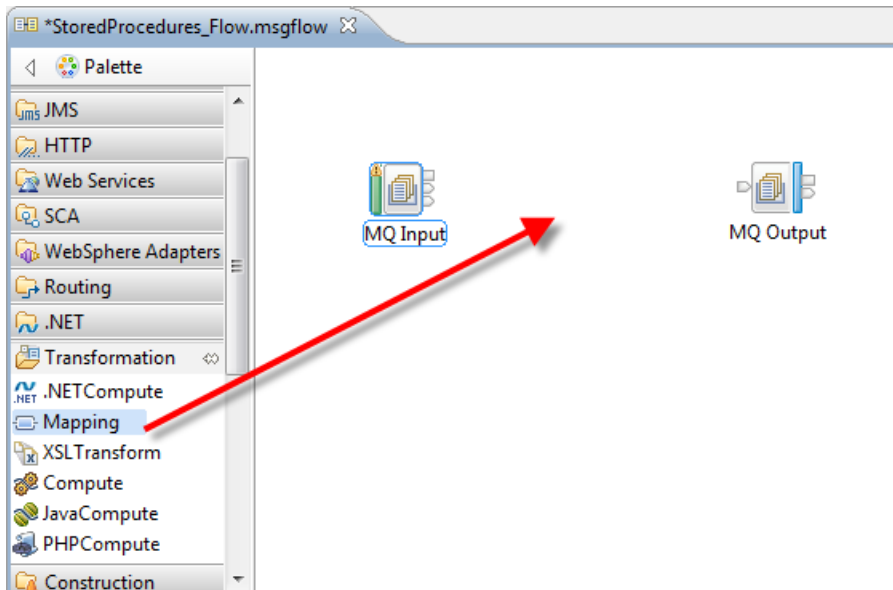
__15. Once the flow is opened, single click on the **MQ Input** node. Click on the **Basic** tab in **Properties** to see its queue. You may have to click the **Properties** tab if it is not already selected.



__16. Click on the **Input Message Parsing** tab and for **Message domain** select **XMLNSC**.



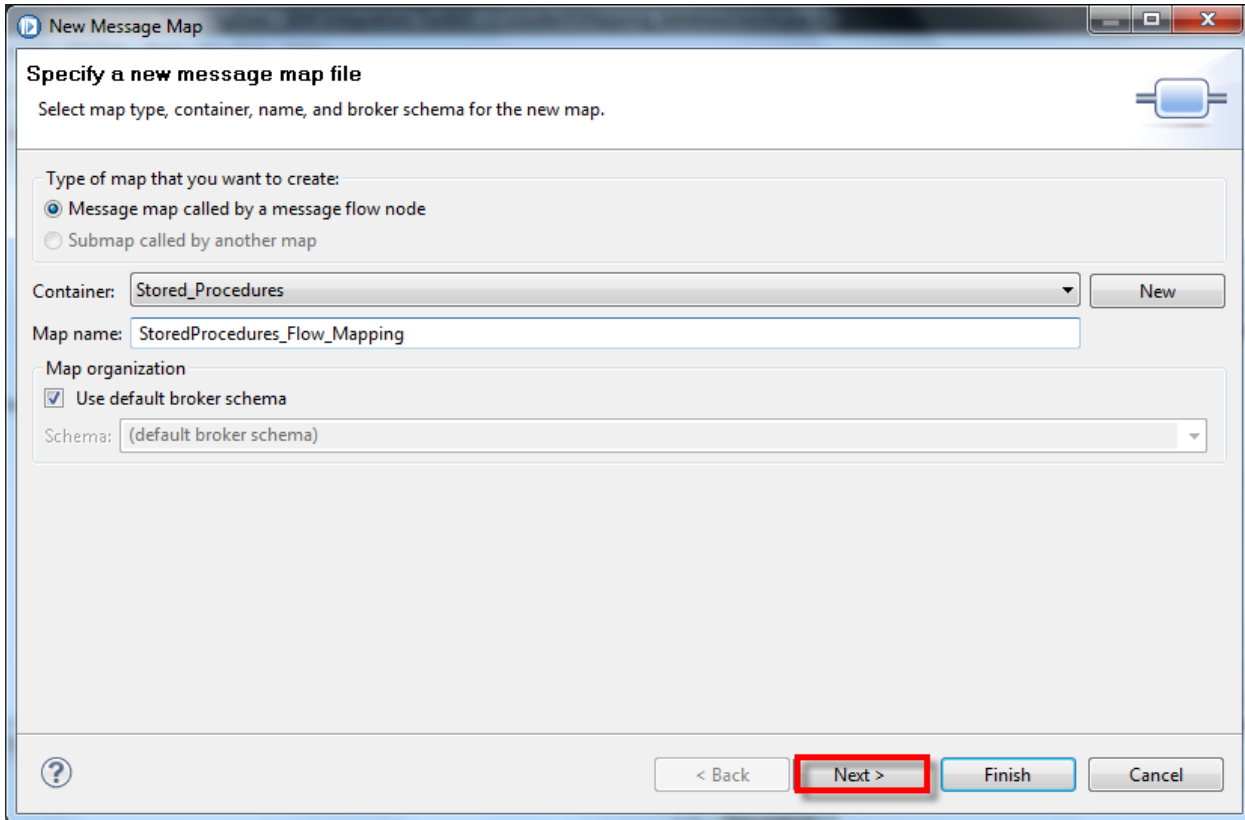
__17. Drag and drop a **Mapping** node, from the Transformation folder, into the message flow between the two MQ nodes.



__18. Double-click on the **Mapping** node.



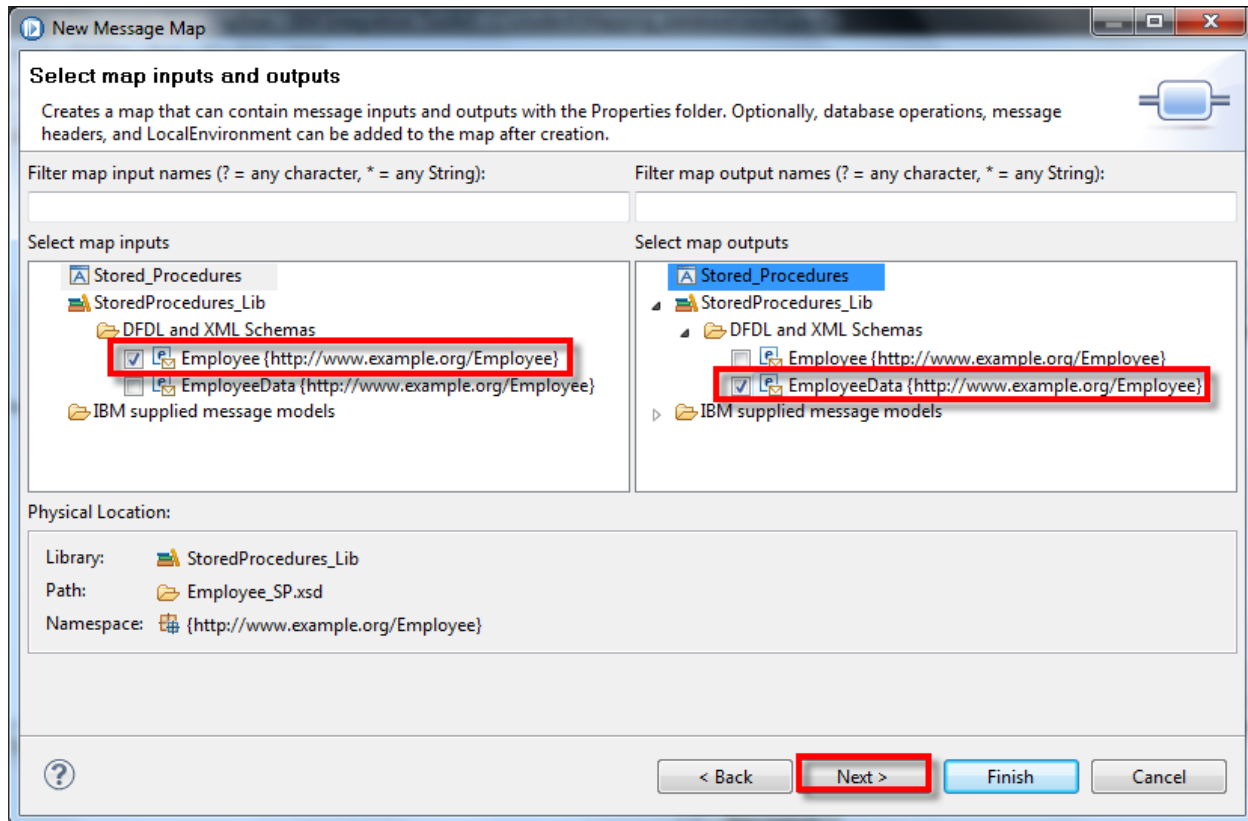
___19. Keep the default values for the message map and click **Next**.



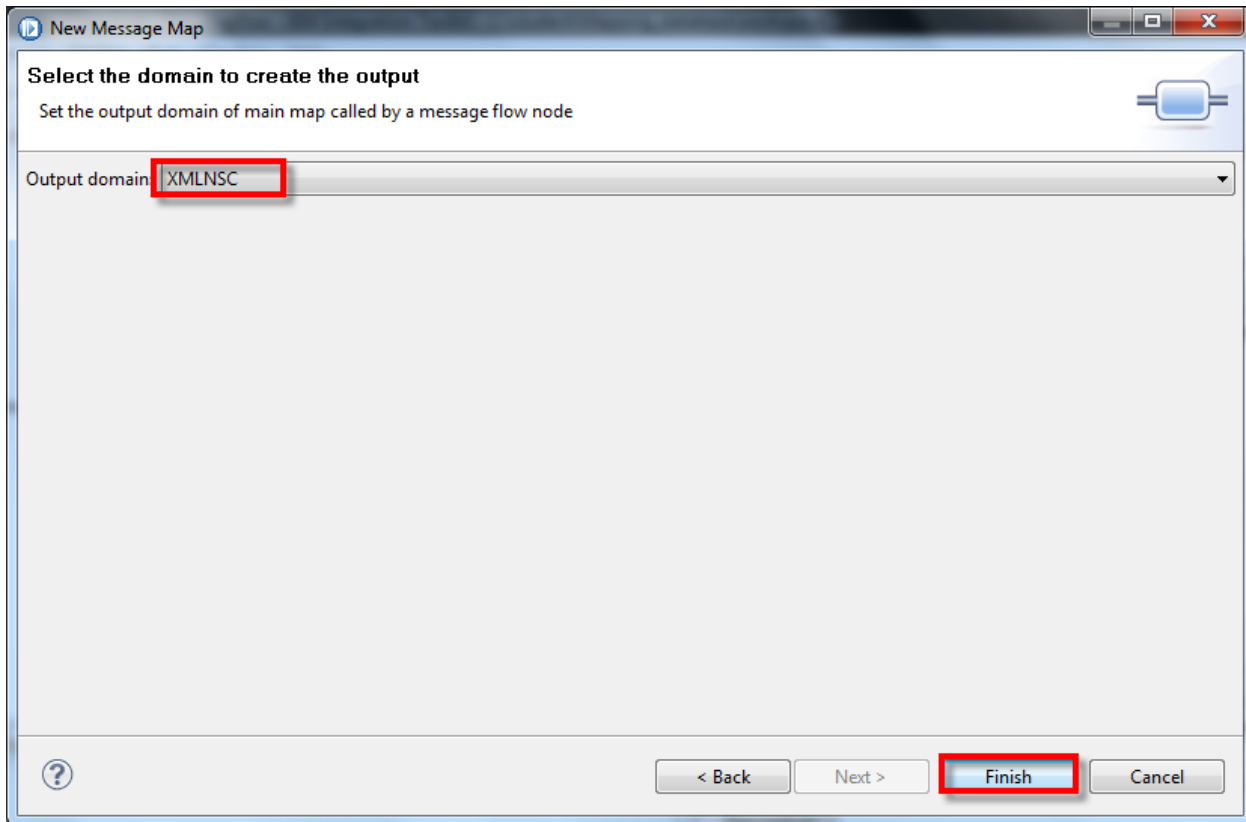
The image shows a 'New Message Map' dialog box with the following fields and controls:

- Title Bar:** New Message Map
- Section Header:** Specify a new message map file
- Instruction:** Select map type, container, name, and broker schema for the new map.
- Type of map that you want to create:**
 - ☒ Message map called by a message flow node
 - ☐ Submap called by another map
- Container:** Stored_Procedures (dropdown menu)
- Map name:** StoredProcedures_Flow_Mapping (text field)
- Map organization:**
 - ☒ Use default broker schema
 - Schema:** (default broker schema) (dropdown menu)
- Buttons:** < Back, Next > (highlighted with a red rectangle), Finish, Cancel

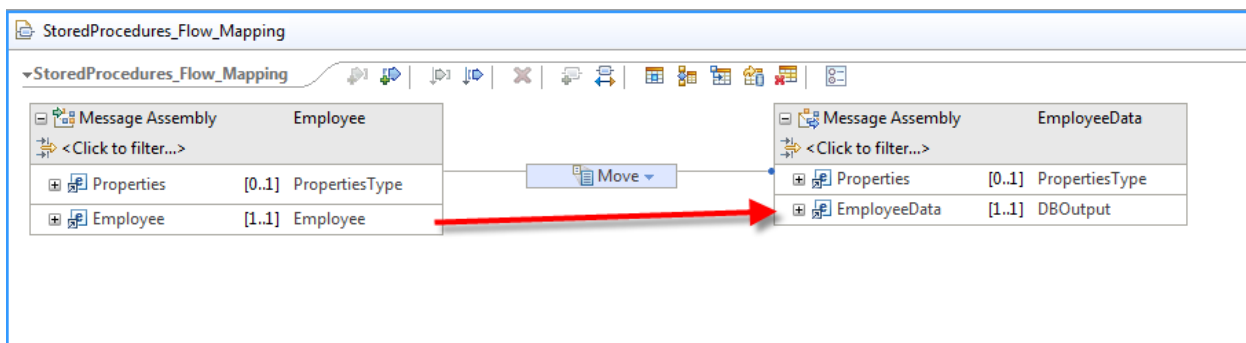
__20. Expand **StoredProcedures_lib** for map inputs and map outputs. Select the **Employee** schema as the map input, and select the **EmployeeData** schema as the map output. Click **Next**.



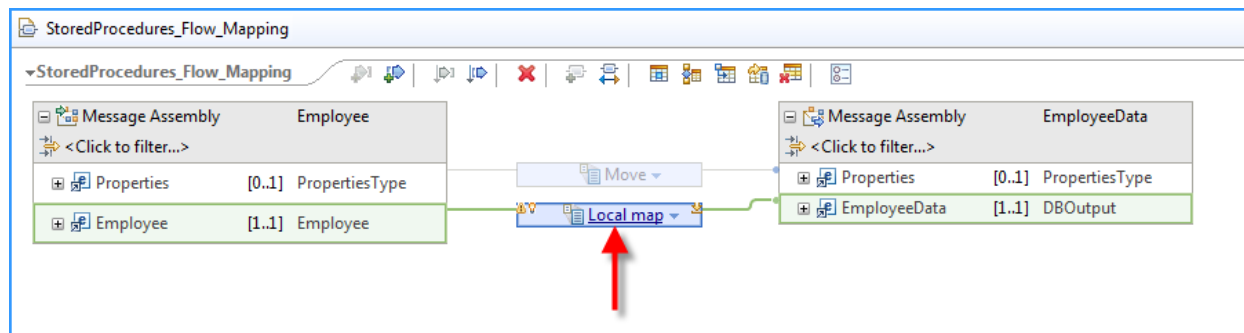
__21. On the next screen click **Finish**, ensuring that **XMLNSC** is selected as the output domain.



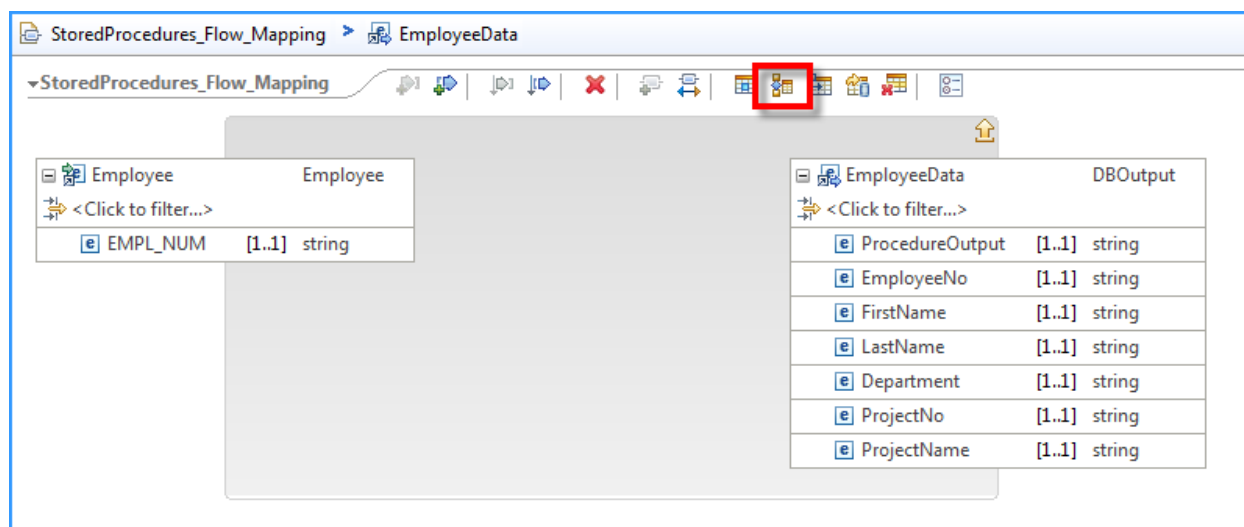
__22. In the opened map, connect **Employee** with **EmployeeData** by hovering over **Employee**. When you see the connector, grab and drop it over **EmployeeData**.



23. The mapper creates a Local map for the operation since you are mapping an entire structure. Click on **Local map**.

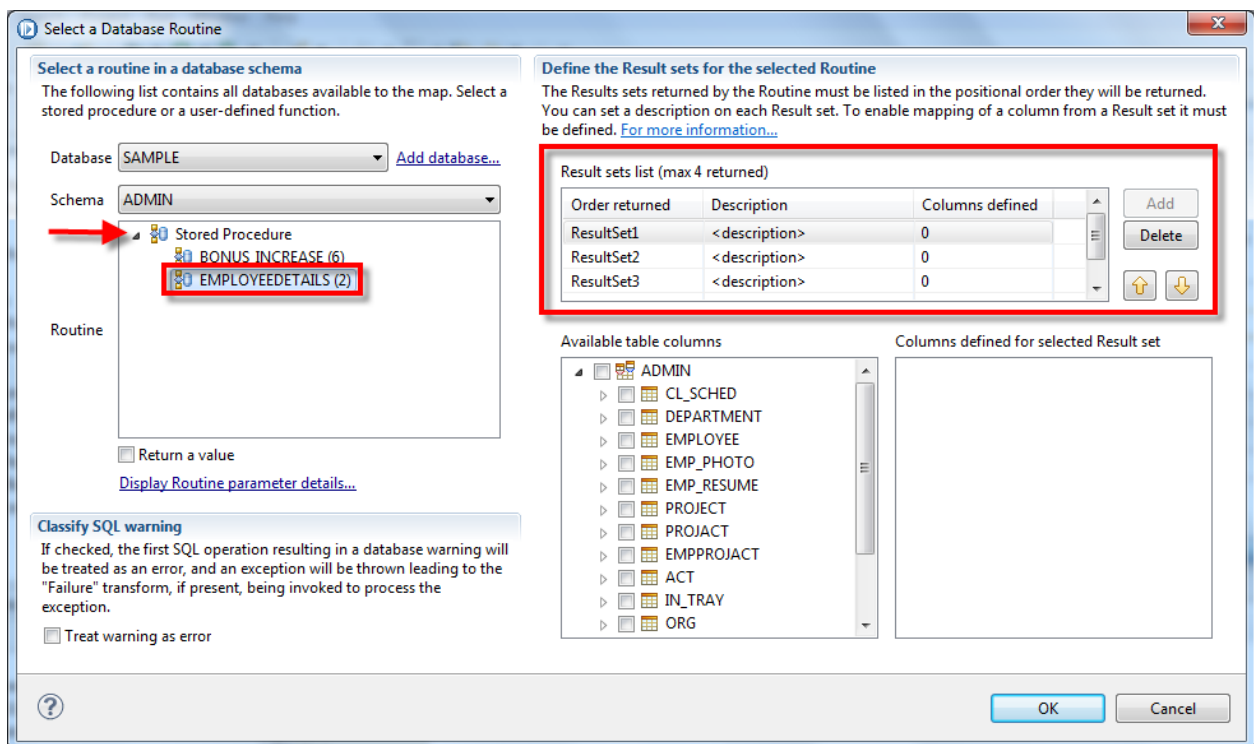


24. In the local map, click on the **Call** routine icon. If not sure, hover over the icons and you should see **Call a database routine**.



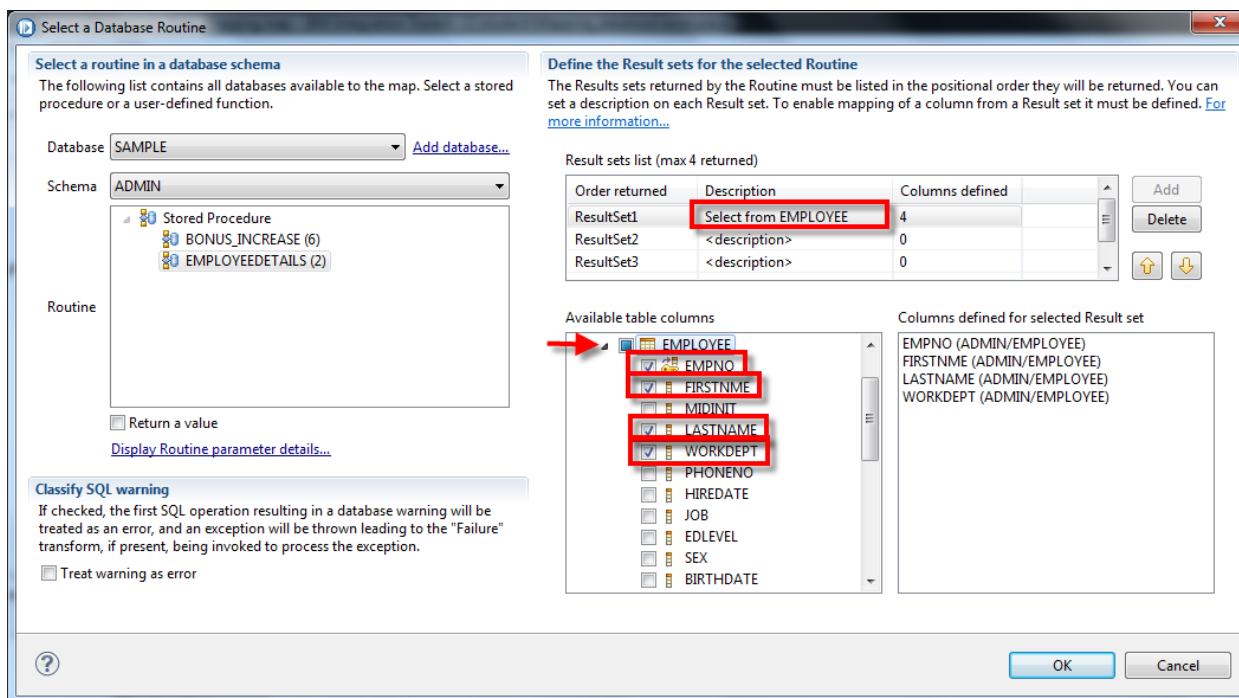
25. In the opened editor, expand **Stored Procedure**, click on **EMPLOYEEDETAILS(2)** and observe the following:

- The 4 result sets (cursors) that we defined in our procedure have been populated in the **Result sets list**.
- For each Result set, you will have to select the table columns that will be defined in the set.
- You can order Result sets at your preference.



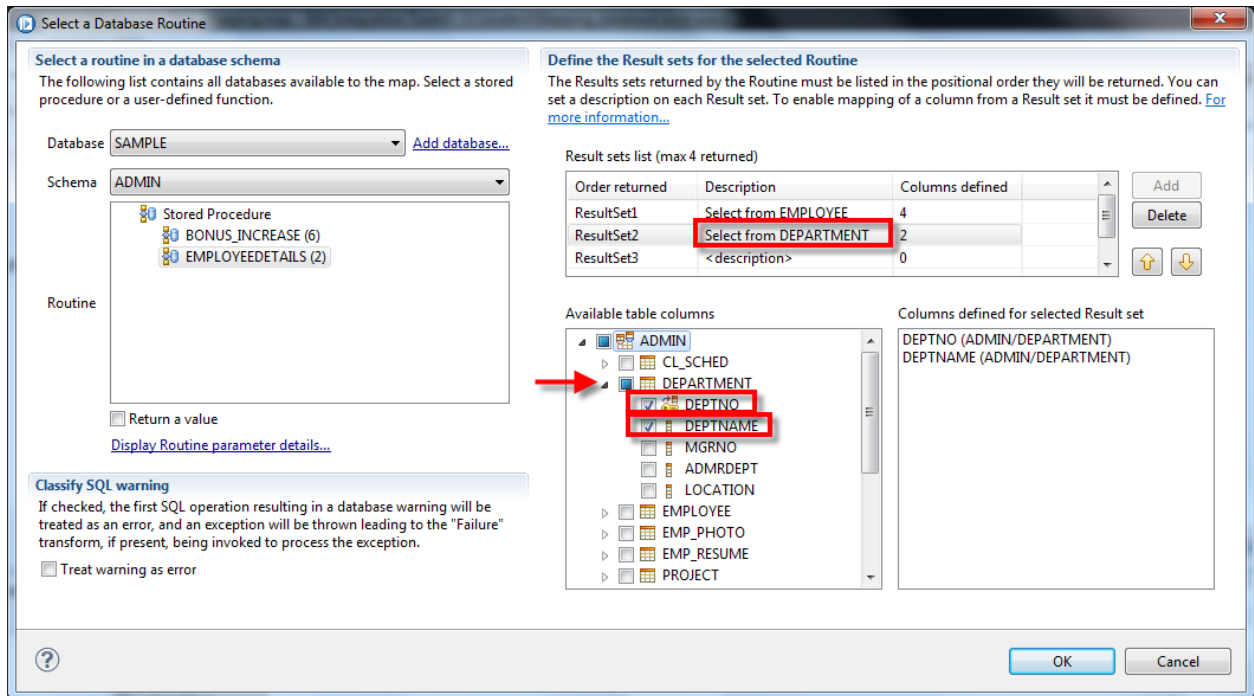
__26. We will start with the **ResultSet1**, and based on our procedure, you will define the following:

- Click on the row with **ResultSet1**. Expand the **EMPLOYEE** table and select the columns **EMPNO**, **FIRSTNAME**, **LASTNAME** and **WORKDEPT**.
- Click on **<description>** and enter **Select from EMPLOYEE**.



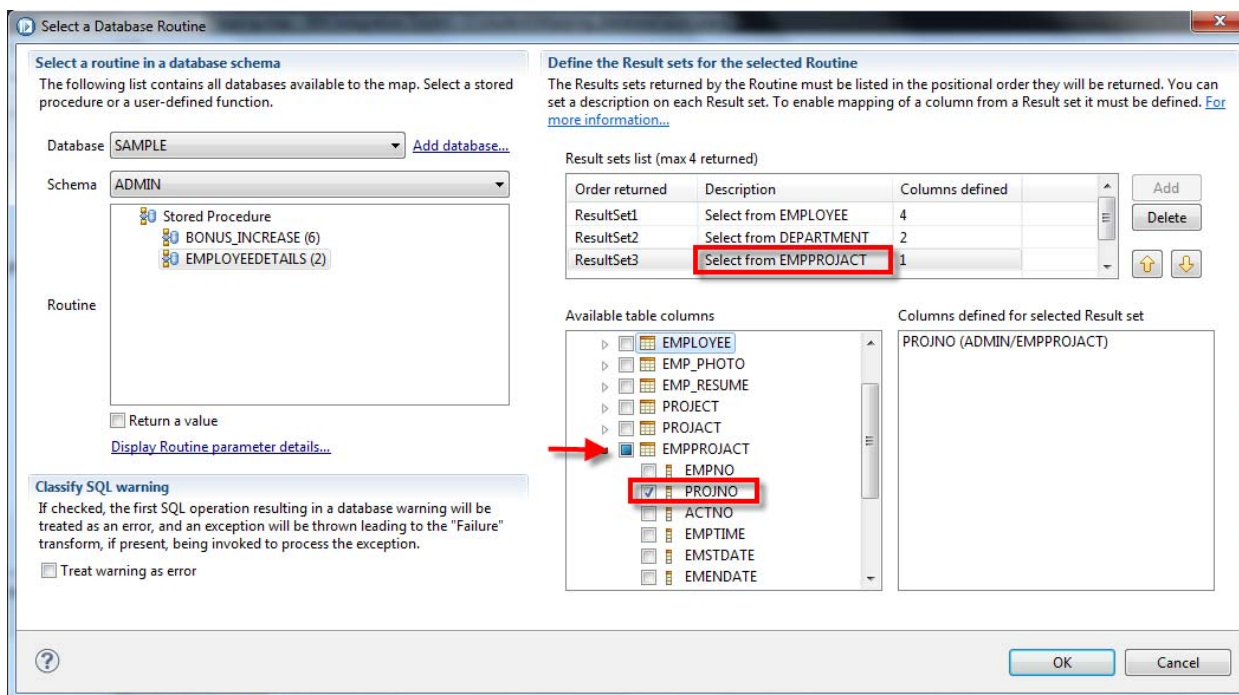
__27. For **ResultSet2**, define the following:

- Click on the row with **ResultSet2**. Expand the **DEPARTMENT** table and select the columns **DEPTNO** and **DEPTNAME**.
- Click on **<description>** and enter **Select from DEPARTMENT**.



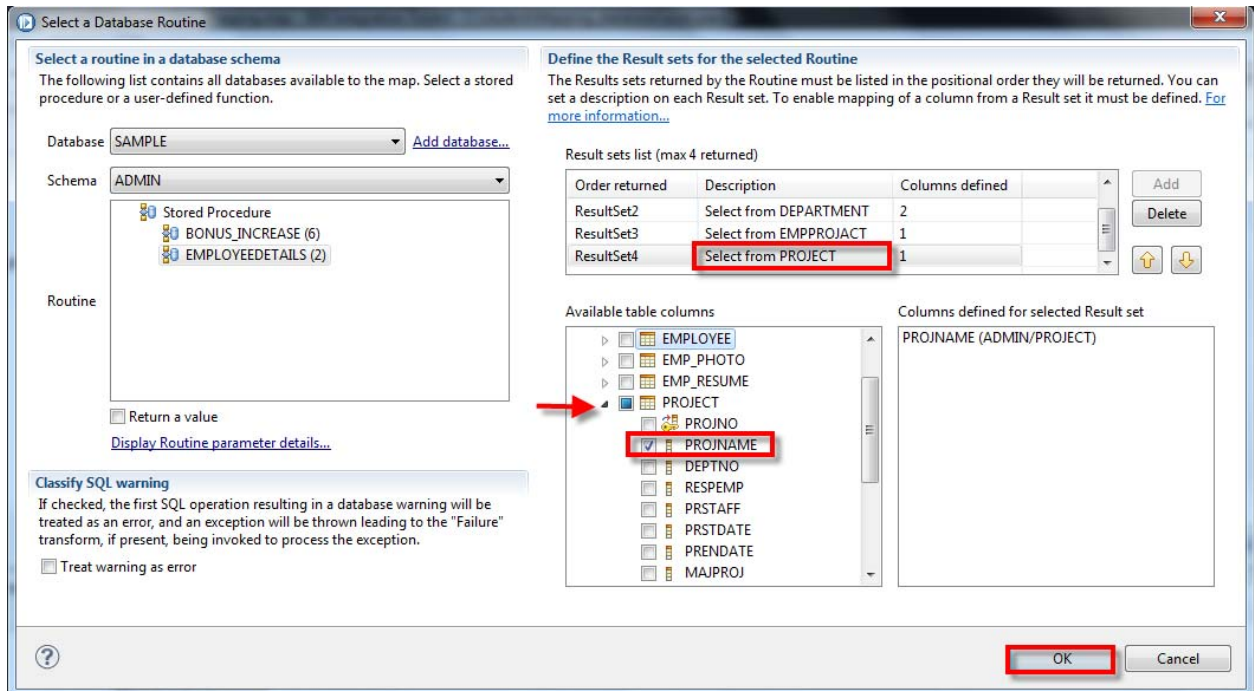
__28. For **ResultSet3** define the following:

- Click on the row with **ResultSet3**. Expand the **EMPPROJECT** table and select the column **PROJNO**.
- Click on **<description>** and enter **Select from EMPPROJECT**.



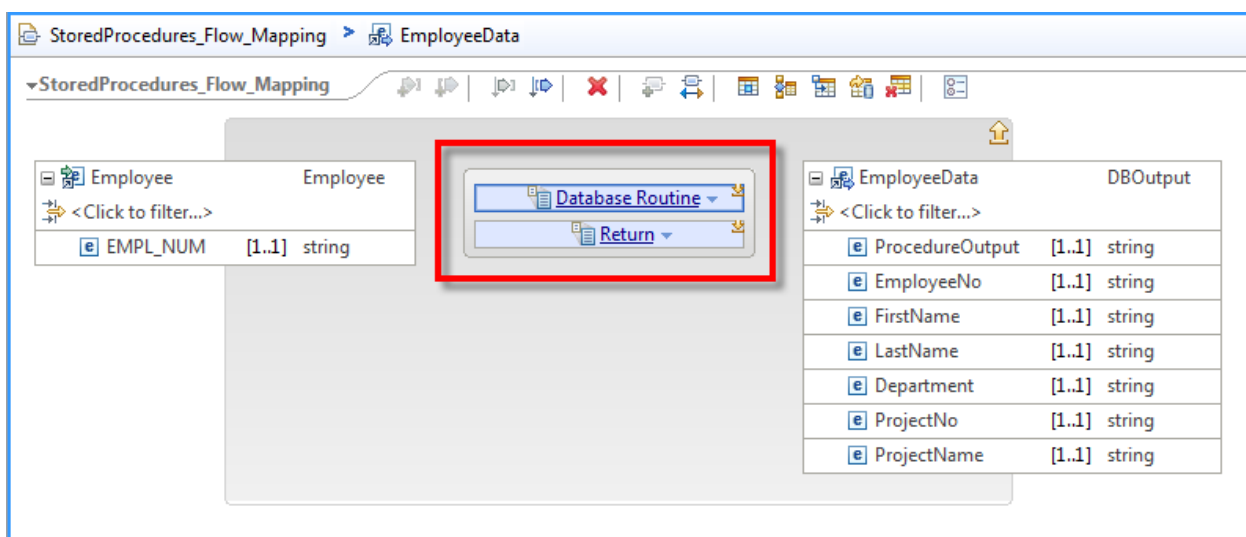
__29. For **ResultSet4** define the following:

- Click on the row with **ResultSet4**. Expand the **PROJECT** table and select the column **PROJNAME**.
- Click on **<description>** and enter **Select from PROJECT**.

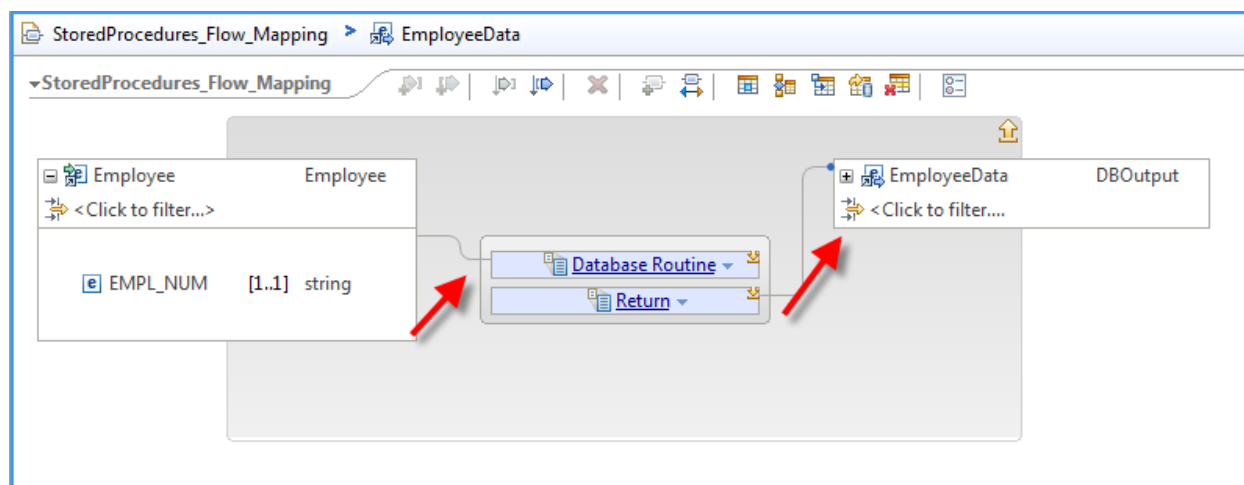


__30. Click **OK**.

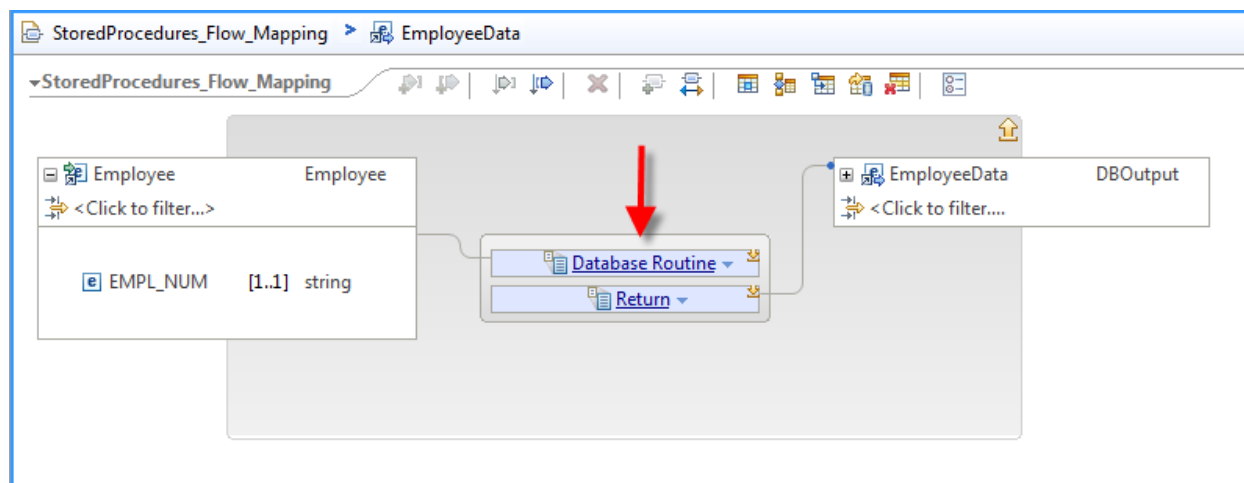
__31. This will return the map with **Database Routine** and **Return** sub maps.



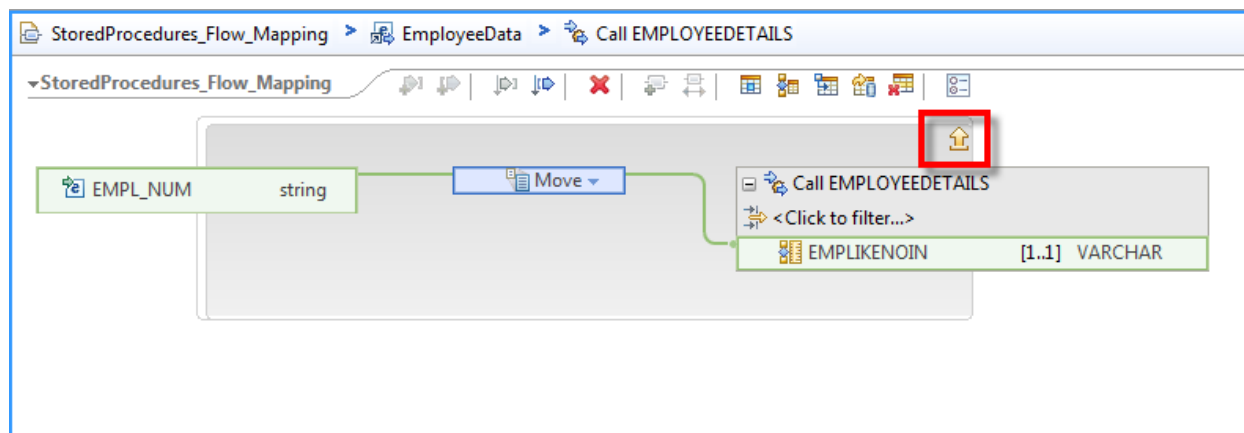
__32. Connect **EMPL_NUM** to **Database Routine** and **Return** to **EmployeeData**.



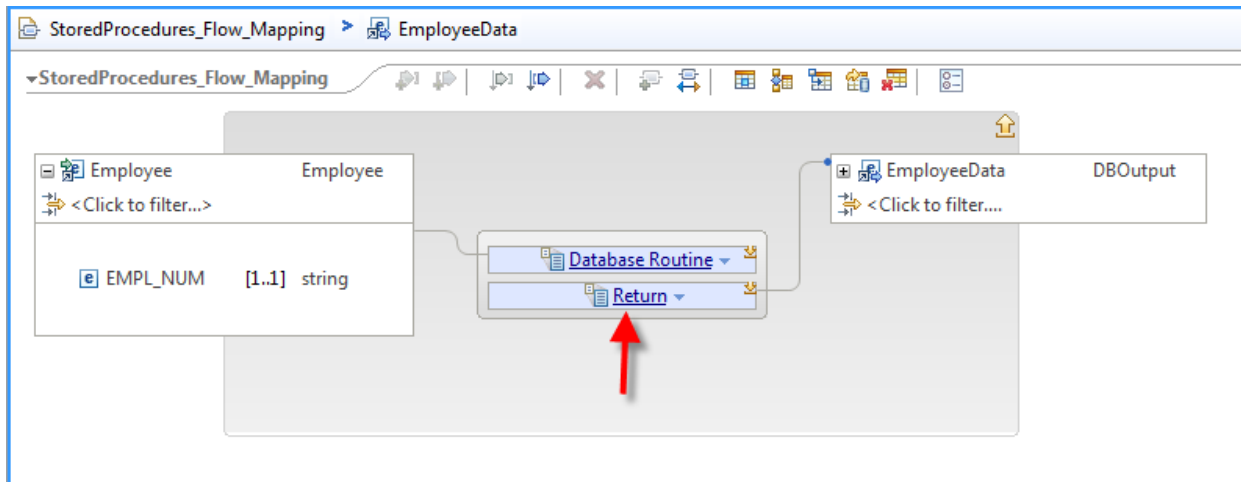
__33. Click on the **Database Routine** link.



__34. In the Database Routine map, connect **EMPL_NUM** to **EMPLIKENOIN** (the input parameter for the routine). Click on the **yellow arrow** to go one level up in the map.



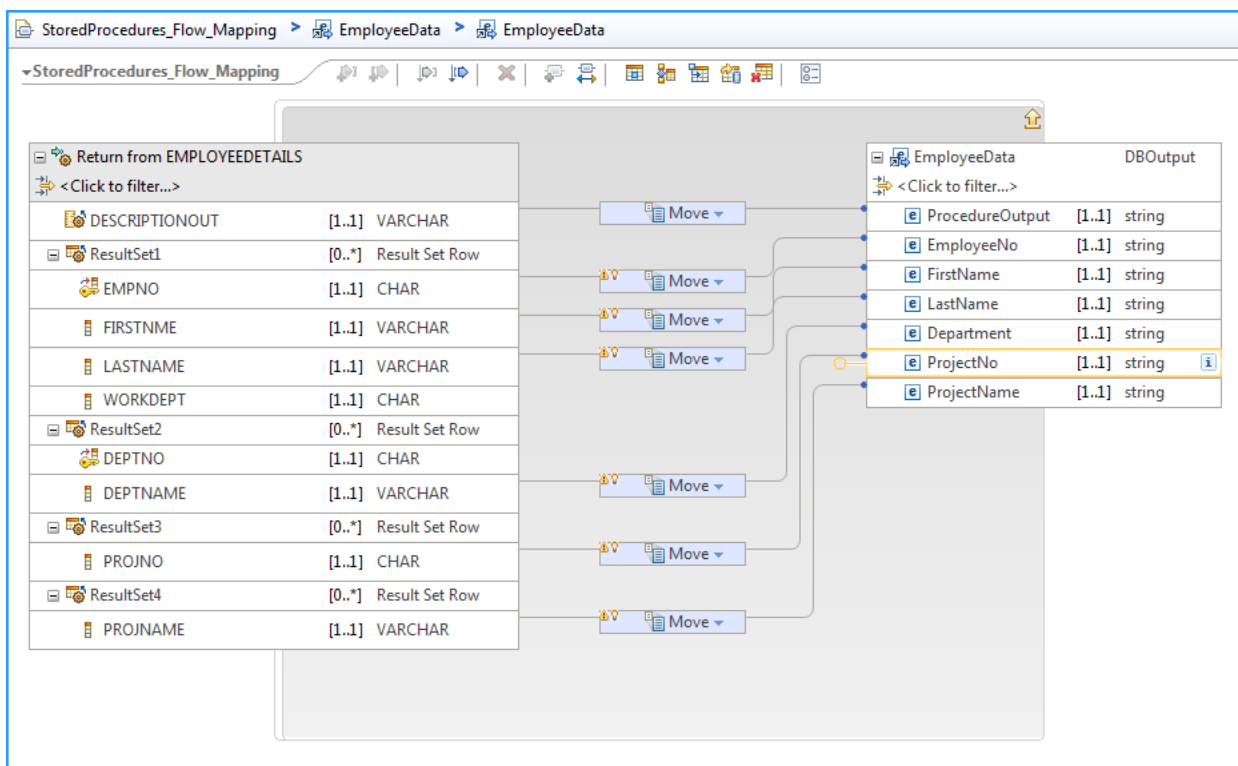
__35. Click on the **Return** sub map link.



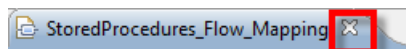
__36. Expand the four result sets. You will see that they have the columns that you selected in steps 26 – 29. Make the following mappings:

- **DESCRIPTIONOUT** → **ProcedureOutput**
- **EMPNO** → **EmployeeNo**
- **FIRSTNAME** → **FirstName**
- **LASTNAME** → **LastName**
- **DEPTNAME** → **Department**
- **PROJNO** → **ProjectNo**
- **PROJNAME** → **ProjectName**

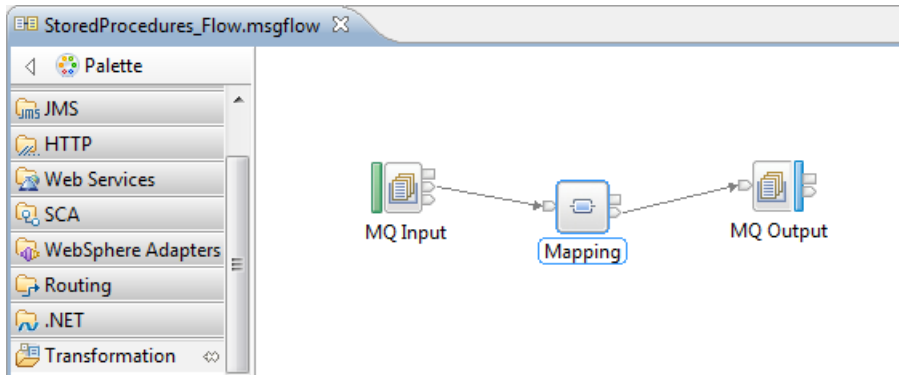
Once finished your map should look like the screen capture below:



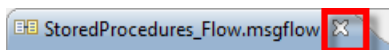
__37. Save the map with **Ctrl+S** and close it.



__38. Connect the nodes in the flow as shown, and save it with **Ctrl+S**.



__39. Close the message flow.

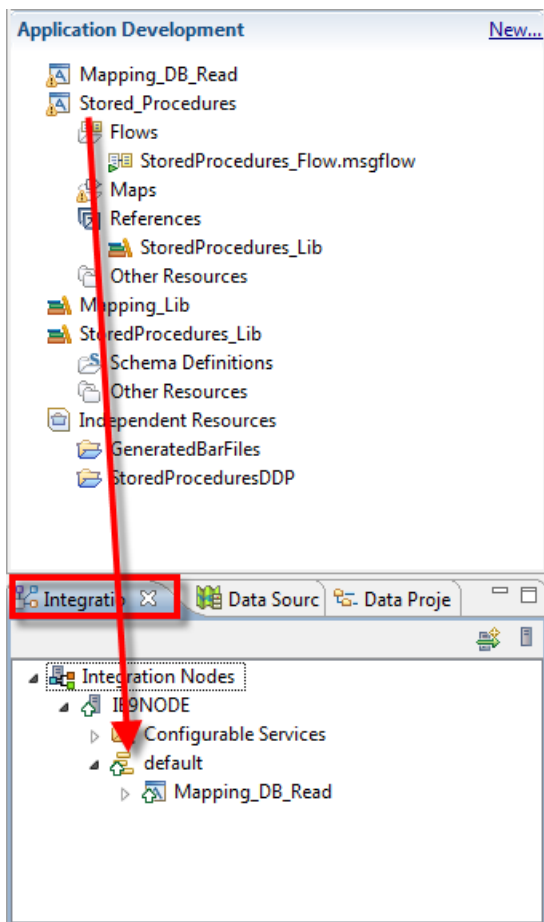


3.4.3 Deploy and test application

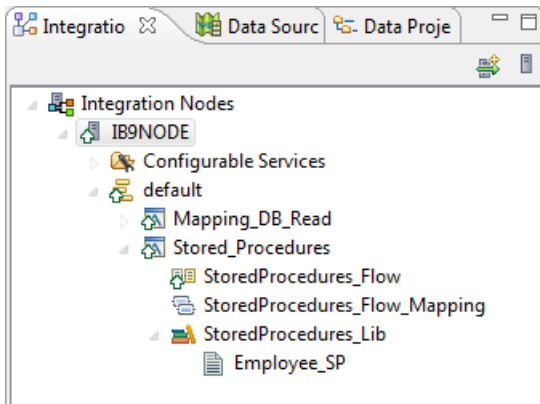
You will now be deploying and testing the created application.

Please note that the Integration Node requires a Configurable Service for the JDBC connection to the database. This is already created. The steps to create it are documented in the Appendix to this lab.

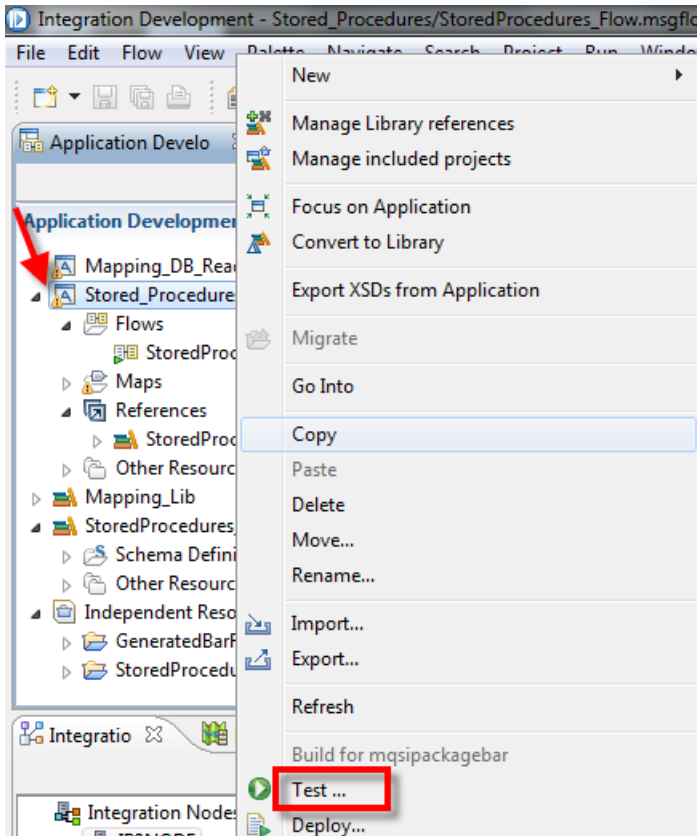
- ___1. Click on the **Integration Node** tab to view the running node. Deploy the **Stored_Procedures** application to the **default** Integration Server.



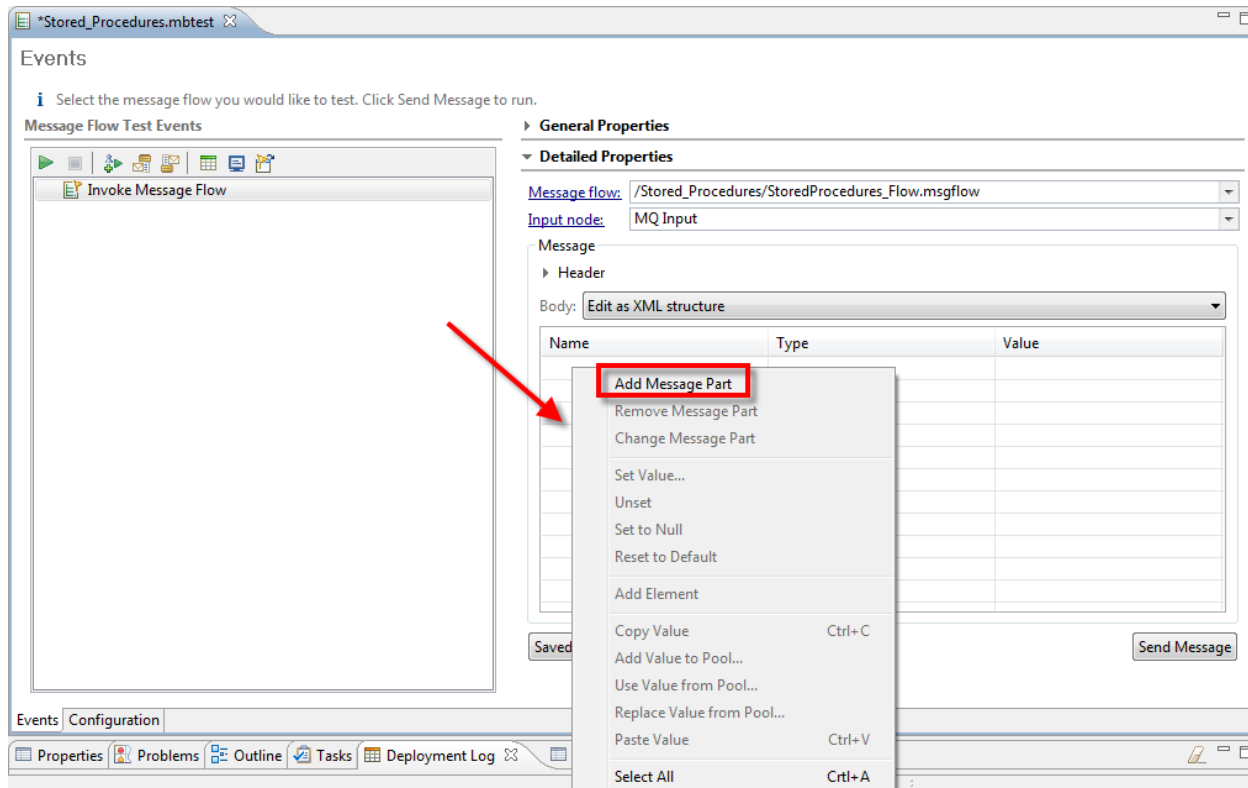
__2. Confirm the deployment by expanding the **default** Integration Server.



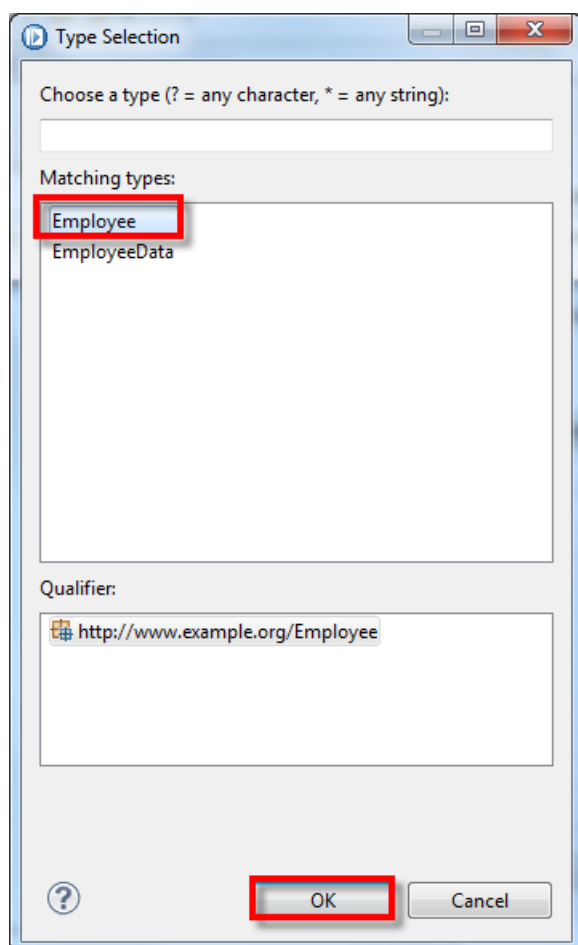
__3. Now that the application is deployed, you will use the Integration Bus embedded Test Client. Right-click on the **Stored_Procedures** application and from the drop-down menu select **Test...**



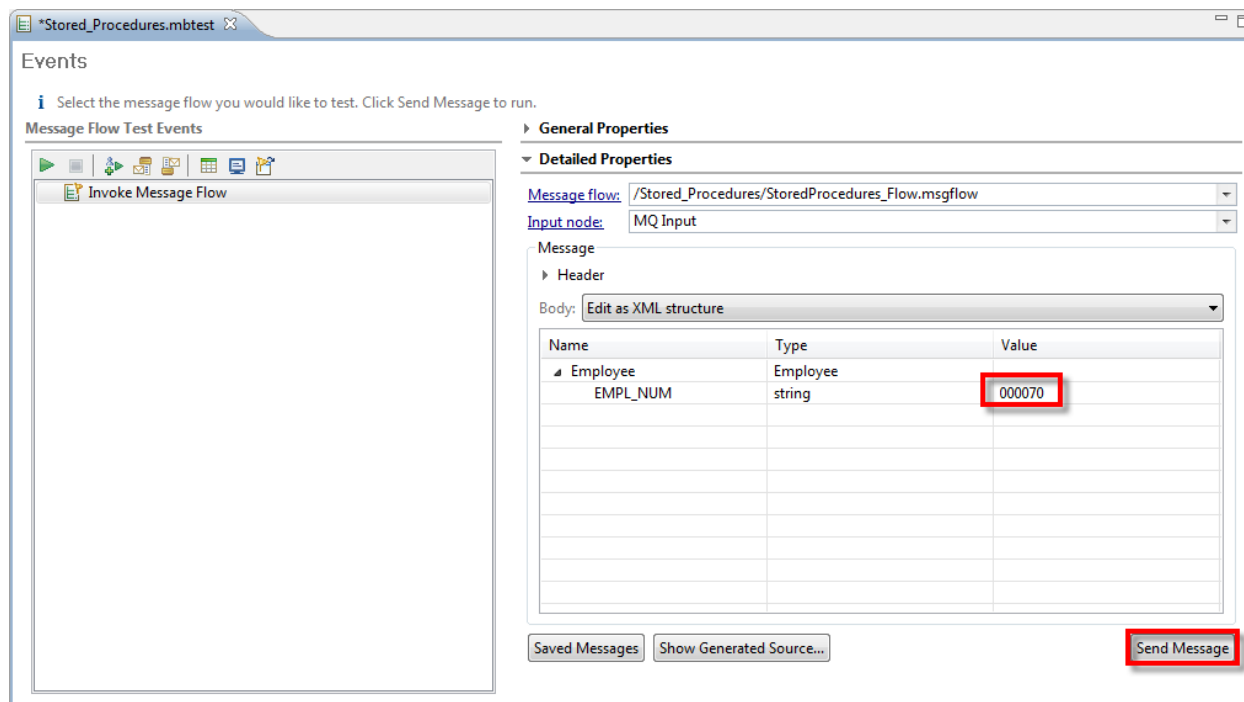
4. In the opened Test client, right-click on the **Body** table and click **Add Message Part**.



__5. From the Type Selection, select **Employee** and click **OK**.

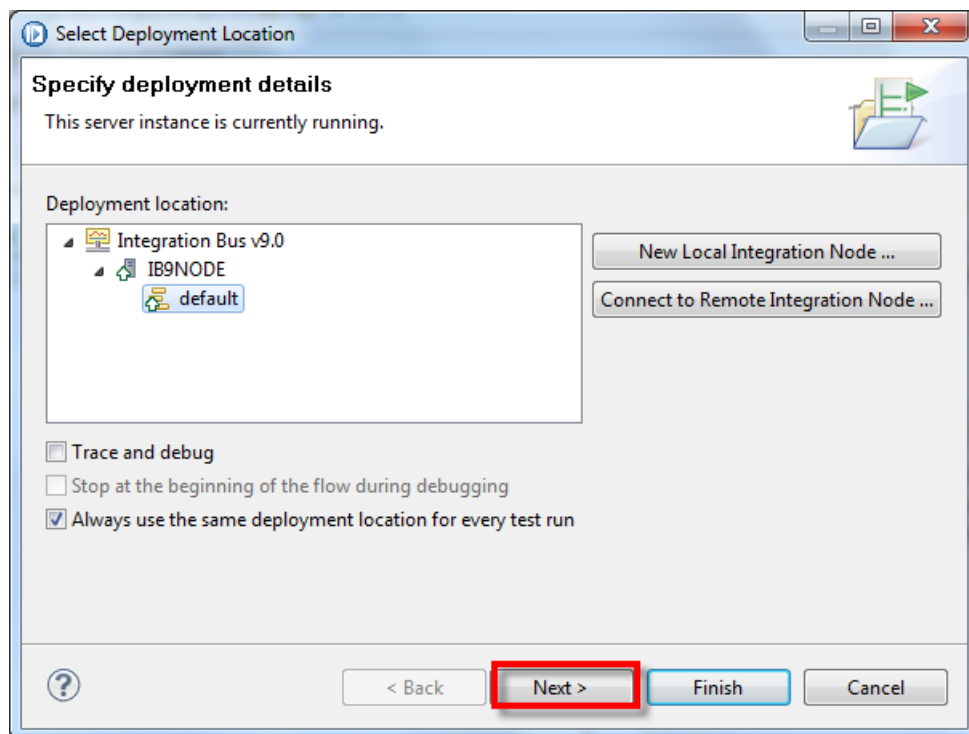


__6. The message body will open as an XML structure. In the **Value** column, change the default value (**EMP_NUM**) to the number '000070'. Once done, it should appear as follows:

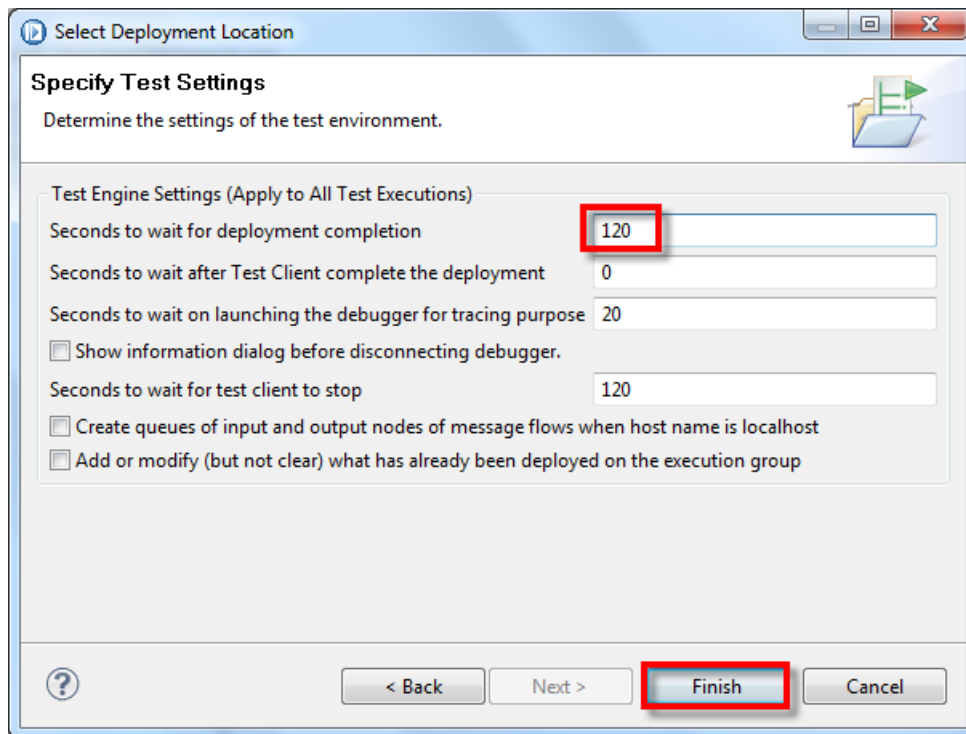


__7. Click **Send Message**.

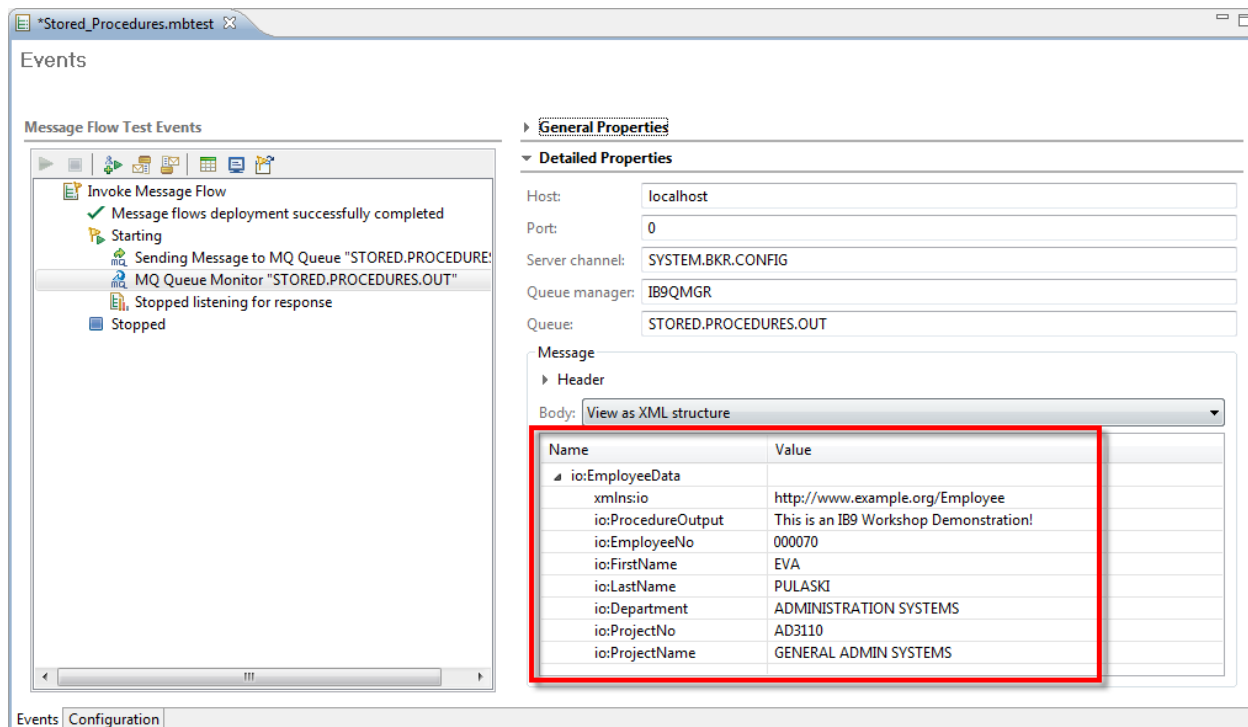
__8. Confirm the deployment to the **default** Integration Server and click **Next**.



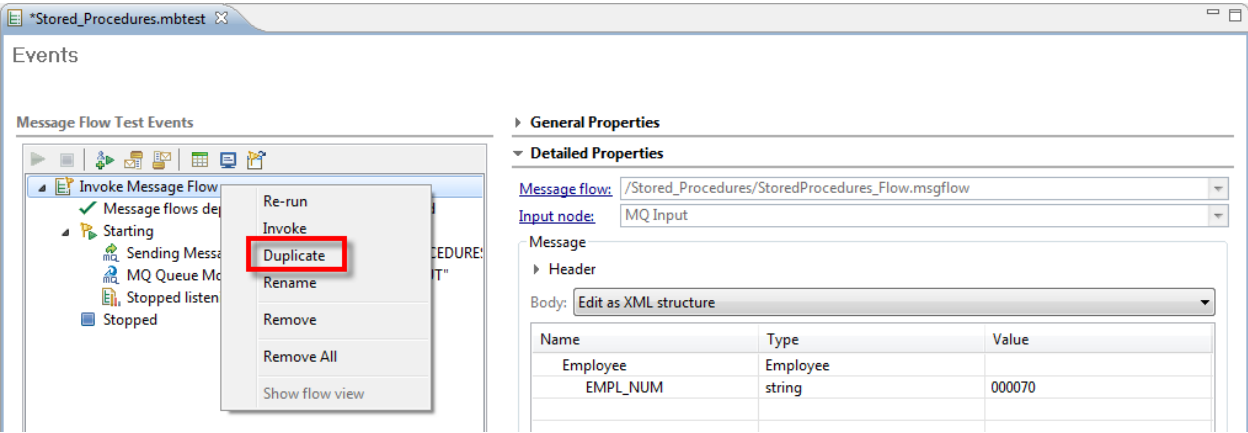
9. On the next screen, increase the default wait time for deployment to **120** as shown. Click **Finish**.



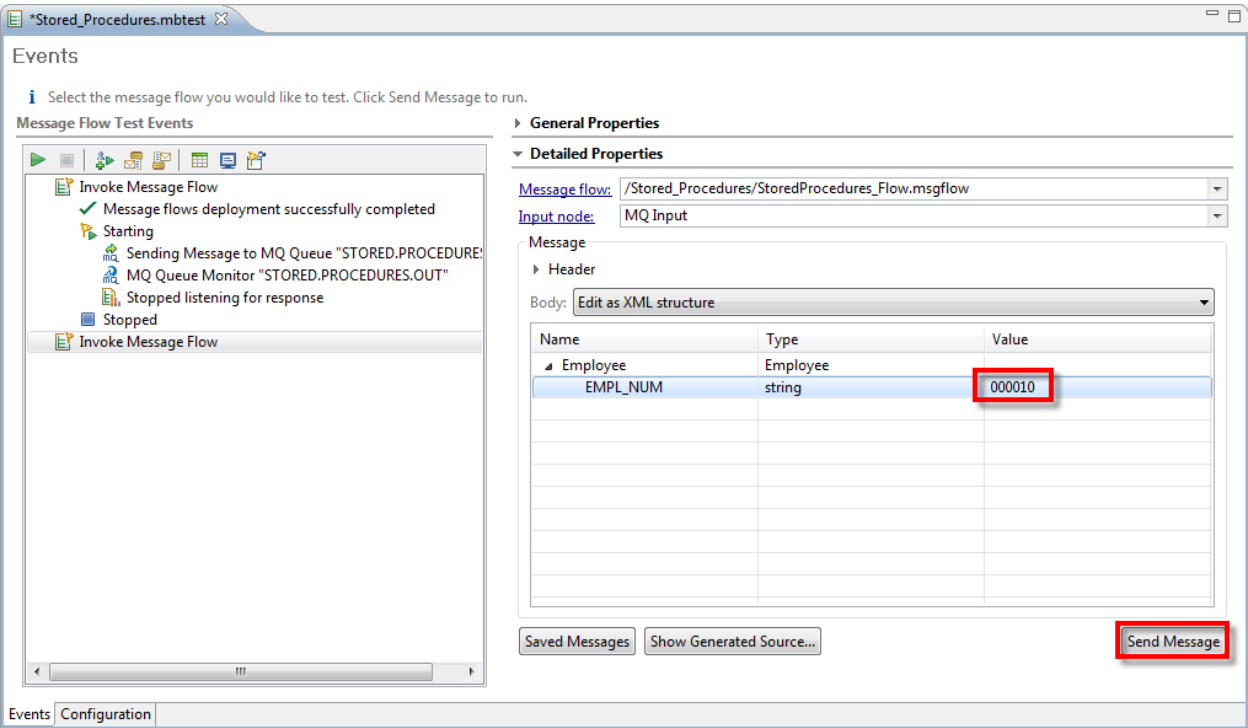
10. It may take a couple of seconds, and once the call is completed you should see the response from the database with the details for the employee with number '000070'.



__11. Feel free to run the test again using different employee number. Right-click on **Invoke Message Flow** and click on **Duplicate**.



__12. In the **Value** column put the number '000010' and click on **Send Message**.



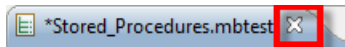
__13. You will receive the response for this employee.

The screenshot displays the IBM Message Flow Test Client window titled "*Stored_Procedures.mbttest". The interface is divided into three main sections:

- Events:** A list of test events on the left. The first event, "Invoke Message Flow", shows a successful deployment and message sending to the "STORED.PROCEDURES.OUT" queue. The second event, "Invoke Message Flow", shows a successful deployment and message sending to the "STORED.PROCEDURES.OUT" queue.
- General Properties:** A section on the right showing the test configuration:
 - Host: localhost
 - Port: 0
 - Server channel: SYSTEM.BKR.CONFIG
 - Queue manager: IB9QMGR
 - Queue: STORED.PROCEDURES.OUT
- Detailed Properties:** A section on the right showing the message body. The "Body" is set to "View as XML structure". A table lists the message properties:

Name	Value
io:EmployeeData	
xmlns:io	http://www.example.org/Employee
io:ProcedureOutput	This is an IB9 Workshop Demonstration!
io:EmployeeNo	000010
io:FirstName	CHRISTINE
io:LastName	HAAS
io:Department	SPIFFY COMPUTER SERVICE DIV.
io:ProjectNo	AD3100 MA2100 MA2110
io:ProjectName	ADMIN SERVICES

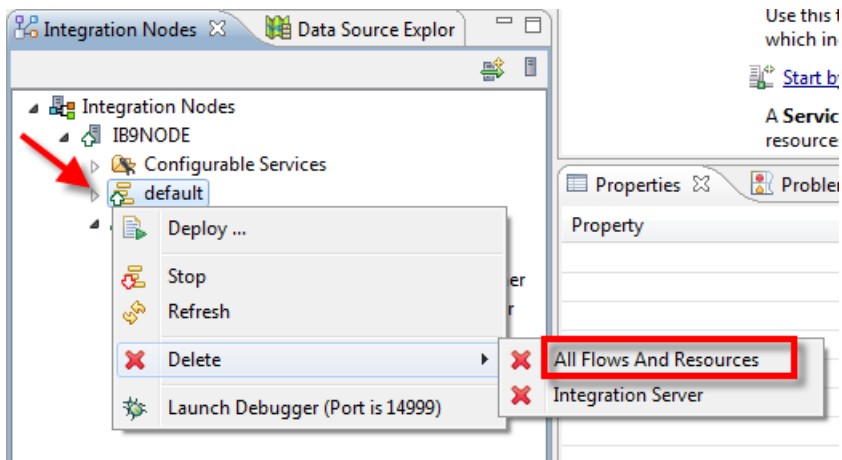
__14. Close the Test Client. Do not save it.



3.5 Clean up

We will remove all of the assets that we have deployed.

1. Remove the deployed applications. **Right-click** on the **default** Integration Server and select **Delete→All Flows and Resources**.



This completes lab 3.

3.6 Appendix - JDBC configuration

The Mapping Node uses JDBC to connect to a relational database. The following commands and configuration are included for reference. This configuration has already been done on the VMWare image, so you do not need to execute the command in this section.

- __1. Set the security parameters to allow the Integration Node to connect to the SAMPLE database.

```
mqsisetdbparms IB9NODE -n jdbc::SAMPLE -u admin -p oneclick
```

- __2. Specify the JDBC security parameters to allow a message flow using the SAMPLE database to connect successfully.

```
mqsisetdbparms IB9NODE -n jdbc::mySecurityId -u admin -p oneclick
```


- __3. Create the configurable service which will use the “mySecurityId” database connection (the command is saved in c:\student\Mapping_database\install\DBSetup\createJDBCCConnections.cmd):

```
mqsicreateconfigurableservice IB9NODE -c JDBCProviders -o SAMPLE -n
connectionUrlFormat,connectionUrlFormatAttr1,connectionUrlFormatAttr2,c
onnectionUrlFormatAttr3,connectionUrlFormatAttr4,connectionUrlFormatAtt
r5,databaseName,databaseType,databaseVersion,description,environmentPar
ms,jarsURL,jdbcProviderXASupport,maxConnectionPoolSize,portNumber,secur
ityIdentity,serverName,type4DatasourceClassName,type4DriverClassName -
v
"jdbc:db2://[serverName]:[portNumber]/[databaseName]:user=[user];passwo
rd=[password];","","","","","","","SAMPLE","DB2 Universal
Database","10.1","default_Description","default_none","C:\IBM\SQLLIB\ja
va","true","0","50000","mySecurityId","localhost","com.ibm.db2.jcc.DB2X
ADataSource","com.ibm.db2.jcc.DB2Driver"
```

Alternatively, you can create it in IB Explorer.

At a minimum, the following parameters should be changed. This example is for DB2 (local instance).

Name:	SAMPLE	
Type:	JDBCProviders	
Template:	DB2	
databaseName:	SAMPLE	
jarsURL:	c:\IBM\SQLLIB\java (or wherever DB2 has been installed)	
securityIdentity:	mySecurityId	
serverName	localhost	

 JDBCProviders/SAMPLE - Properties

Configurable Service

Modify a Configurable Service's attributes

*Name	SAMPLE
*Type	JDBCProviders
Template	SAMPLE

Key	Value
connectionUrlFormatAttr5	
databaseName	SAMPLE
databaseSchemaNames	useProvidedSchemaNames
databaseType	DB2 Universal Database
databaseVersion	10.1
description	default_Description
environmentParms	default_none
jarsURL	C:\IBM\SQLLIB\java
jdbcProviderXASupport	true
maxConnectionPoolSize	0
portNumber	50000
securityIdentity	mySecurityId
serverName	localhost
type4DatasourceClassNa...	com.ibm.db2.jcc.DB2XADataSource
type4DriverClassName	com.ibm.db2.jcc.DB2Driver

__4. This will result in a new configurable service as shown below:

The screenshot shows the MQ Explorer interface. On the left, the 'MQ Explorer - Navigator' pane displays a tree view of the system. Under 'Integration Nodes' > 'IB9NODE' > 'default', the 'Configurable Services' folder is expanded, showing a new service named 'JDBCProviders/SAMPLE'. On the right, the 'MQ Explorer - Content' pane shows the 'Configurable Service SAMPLE' details. Below the title, a 'Properties QuickView' table lists various configuration parameters.

Properties QuickView:	
Name	SAMPLE
Type	JDBCProviders
connectionUrlFormat	jdbc:db2://[serverName]:[portNumber]/[datab
connectionUrlFormatAttr1	
connectionUrlFormatAttr2	
connectionUrlFormatAttr3	
connectionUrlFormatAttr4	
connectionUrlFormatAttr5	
databaseName	SAMPLE
databaseSchemaNames	useProvidedSchemaNames
databaseType	DB2 Universal Database
databaseVersion	10.1
description	default_Description
environmentParms	default_none
jarsURL	C:\IBM\SQLLIB\java
jdbcProviderXASupport	true
maxConnectionPoolSize	0
portNumber	50000
securityIdentity	mySecurityId
serverName	localhost
type4DatasourceClassName	com.ibm.db2.jcc.DB2XADataSource
type4DriverClassName	com.ibm.db2.jcc.DB2Driver

__5. Stop and restart the default execution group, or the Integration Node.

```
mqsisstop IB9NODE
```

```
mqsisstart IB9NODE
```

Page Intentionally Left Blank

Lab 4 Decision service node

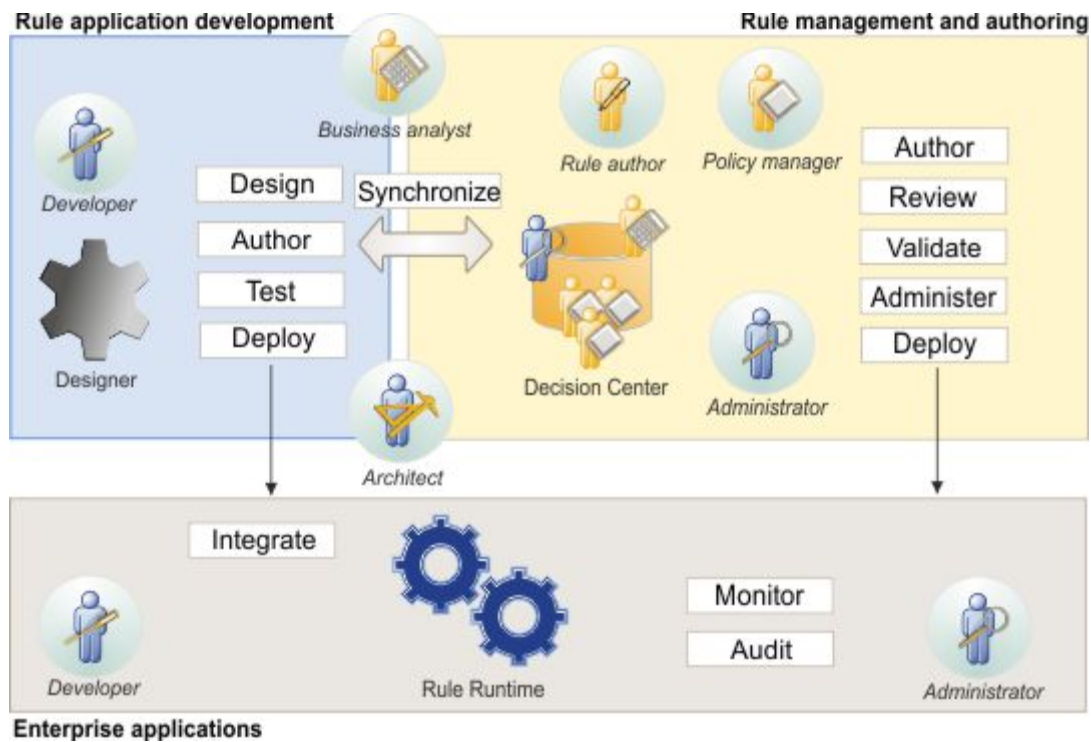
4.1 Introduction to IBM Operational Decision Management

IBM Operational Decision Manager integrates business events and business rules to automate decisions across processes and applications. It improves the quality of transaction and process-related decisions that are made repeatedly, determining the appropriate course of action for each customer, partner and internal interaction.

The IBM Operational Decision Manager modules

IBM Operational Decision Manager comprises a set of modules that operate in different environments, but also work together to provide a comprehensive decision management platform.

- **Rule Designer** – is an integrated development environment (IDE) for rule applications that integrates directly into the Eclipse family of integrated development environments, including Eclipse, IBM Rational Application Developer (RAD) and IBM Rational Software Architect (RSA). Rule Designer is used by developers and architects to develop and integrate business rule sets into applications.
- **Rule Execution Server** – is a robust, J2SE and J2EE-compliant execution environment for deploying business rule SOA services to the leading web and application servers such as WebSphere Application Server. The Rule Execution Server includes components for synchronous, asynchronous and web service-based invocation of business rules and includes a web administration console. The Rule Execution Server is fully integrated with the Rule Designer and Decision Center to support business rule deployment for both developers and policy managers.
- **Decision Center** – A scalable rule management server and repository with a collaborative web environment for authoring, managing, validating and deploying business rules. Decision Center provides project governance, including role-based security, history maintenance and custom metadata. Decision Center provides enhanced collaboration between teams through multi-user access for business users and synchronization between IT and business user environments.
- **Rule Solutions for Office** – is used for offline authoring and sharing of the business rules using standard office tools. Rule authors write rules in Microsoft Word and edit decision tables in Microsoft Excel. They can create mixed rule and non-rule content in a RuleDoc, and retain semantic information together with the actual implementation content of the rules.



Two categories of users are involved in developing and maintaining a decision management solution:

IT users

Architects, developers and administrators develop and maintain the rule application. Developers work with Rule Designer in Eclipse for design, Java development and rule project development. Using Decision Validation Services, they can also test rule sets against real or fictitious scenarios to support and troubleshoot problems found in Decision Center.

Business users

- Business users work with Decision Center to write and maintain business rules, both during application development and after the application is deployed to production.
- Business users can perform end-user testing and simulation in Decision Center. Business analysts can simulate business outcomes, run updated rules against historical data, simulate expected changes in data profiles against existing rules, and analyze aggregate outcomes.
- Policy managers and other business users can use Rule Solutions for Office to author rules, in a familiar environment. RuleDocs are Microsoft Office documents that contain business rules. Business users can publish RuleDocs from Decision Center, edit the RuleDocs in Word or Excel, and update their changes back into Decision Center.
- RuleDocs can be published to any WebDAV server and can be short-lived or managed in a content management system such as SharePoint.

4.2 Rules in IBM Integration Bus – Decision Service node

IBM Integration Bus V9.0 includes a subset of the IBM Operational Decision Management solution. The IBM Integration Bus runtime includes the rules execution server, handling rules only (not business events too). The IBM Integration Bus Toolkit includes a new node, called the Decision Service node, which executes the business rules, and also includes an editor for creation of those business rules.

Alternatively, you can use IBM Operational Decision Management Decision Center to do rule authoring, and use those rule sets with the IBM Integration Bus Decision Service node. You can also use the IBM Integration Bus Decision Service node to call business rules execution on an external IBM Operational Decision Management Rule Execution Server.

Rules in IBM Integration Bus

You can use IBM Integration Bus to write business rules by using natural language, so that they can be read easily by business users (for example, a business analyst). In IBM Integration Bus, you create a decision service, which is a collection of rules that are used to process a message. A Decision Service node executes those business rules to provide operations like routing, validation and transformation. You can either write rules in the IBM Integration Toolkit, or import rules from IBM Operational Decision Management. You can also retrieve rules from an external IBM Operational Decision Management repository.

The Decision Service node allows IBM Integration Bus to call business rules that run on a component of IBM Decision Server that is provided with IBM Integration Bus. The IBM Integration Bus license entitles you to use this component only through the Decision Service node and only for development and functional testing. To use the IBM Decision Server component beyond development and functional test, you must purchase a separate license entitlement for either IBM Operational Decision Management Decision Server (the full external server) or IBM Operational Decision Management Decision Server Rules Edition for Integration Bus (the rules only server included in IBM Integration Bus).

A business rule is a required operation that applies to a specific set of business conditions. For example, you can create a business rule that offers a discount to customers who spend more than a certain amount. The business rule can be modified in the future if the business climate changes and the amount of discount must change.

You can use business rules to update the business logic that is applied to message processing as business conditions change. Here are some examples of the capability that business rules can provide.

Routing

Business rules provide smart, dynamic routing in the message flow that is based on the business content of the message. For example, business rules can control least cost routing: if a customer places an order, business rules can be used to determine which facility is the most appropriate to fulfill that order. This decision can be based on the types of item that are ordered, the location of the customer, and the required speed of delivery.

Validation

Business rules provide message validation that is based on the business content of the message; for example, business rules can provide content validation. If an account number starts with ABC, the next four characters must be numbers. But if the account number begins with XYZ, the account number must contain five numbers.

Transformation

Business rules can be applied to message content to modify the values in that message. For example, you can change the discount that is offered to a customer when certain conditions are met, such as a purchase above a certain amount.

Consider an example of a business rule where customers who spend a lot of money in a single transaction are provided an upgrade.

To create business rules, first specify the vocabulary that is required to express the policy, and then represent the logic of the business policy as "if-then" statements. In IBM Integration Bus, you use a Decision Service editor to specify the parameters that form the vocabulary for writing rules. A typical parameter is shown in the following example.

Name	Type	Verbalization
myCustomer	CustomerType	The customer

If you already have a decision service file, you can drag it on to a Decision Service node or use the node properties to locate it. Alternatively, you can double-click the node or use menu options to create a decision service by using a wizard. When you import a rule application archive from IBM Operational Decision Management, it is converted to a decision service, which you can associate with a Decision Service node. You cannot use the IBM Integration Toolkit to view or edit the rules that you import from IBM Operational Decision Management.

Business rules act on business objects, such as a customer. This business object is represented by an XML schema. IBM Integration Bus supports rules that are created from XMLNSC or DFDL schemas. For decision services that you create in IBM Integration Bus, you write the business rules in the Decision Service editor by using the vocabulary that you define. The following example shows how natural language parameters are used in the Decision Service editor to implement the business policy:

```

if
    the category of the customer is Gold
    and the value of the shopping cart of the customer is more than $1500
then
    set the category of the customer to Platinum

```

Rules are written by using the Business Action Language (BAL).

These parameters and rules are contained in a decision service (.rules) file that can be deployed. When you deploy a BAR file that contains a decision service that you created in the IBM Integration Toolkit, the decision service is compiled into a rule set (.ruleset). Imported rules contain compiled rule sets. In the form of a decision service, the business logic can be called from the message flow as a business rule application. Therefore, changes to the business policy do not require changes to the message flow.

You can write rules in the IBM Integration Toolkit and at run time retrieve them from the local application, library or Integration project. You can also import business rules that were written in IBM Operational Decision Management and retrieve them from an IBM Operational Decision Management repository at run time. A configurable service is provided for you to specify connection details.

Structure of a business rule

A business rule expresses business logic by using a rule syntax that represents natural language.

A business rule typically consists of the following information, in the order specified:

1. Definitions
2. Conditions
3. Actions

Definitions

At the beginning of the rule, you can set parameters that identify business terms by using easily understandable names. You can set these parameters by using the New Decision Service wizard.

When you specify the parameters, either by using the New Decision Service wizard when you create the decision service, or by adding them later in the Decision Service editor, you can then refer to those parameters from all the rules in the .rules file. If you create a parameter in the definitions section of a specific rule, that parameter is available to that rule only.

Here is an example of a definition:

```
definitions

    set minimum_cart_value to $1500
```

Conditions

The conditions section of the rule contains the "if" statements. These statements define the conditions under which actions are completed. If the condition is true, the action is completed. A rule can contain one or more condition statements.

Here is an example of a condition section that contains two condition statements.

```
if

    the customer's category is Gold

    and the value of the customer's shopping cart is more than minimum_cart_value
```

The second condition statement uses the variable that was set in the definitions section. The action is completed if both statements are true.

Actions

The actions section of the rule contains the "then" statements. These statements define the actions that are taken when the conditions that are represented by the "if" statements are met. If the actions section contains more than one action, the actions are taken in the order in which they

are written. You can also include "else" statements in the actions section. These statements define what actions to take if the conditions are not met.

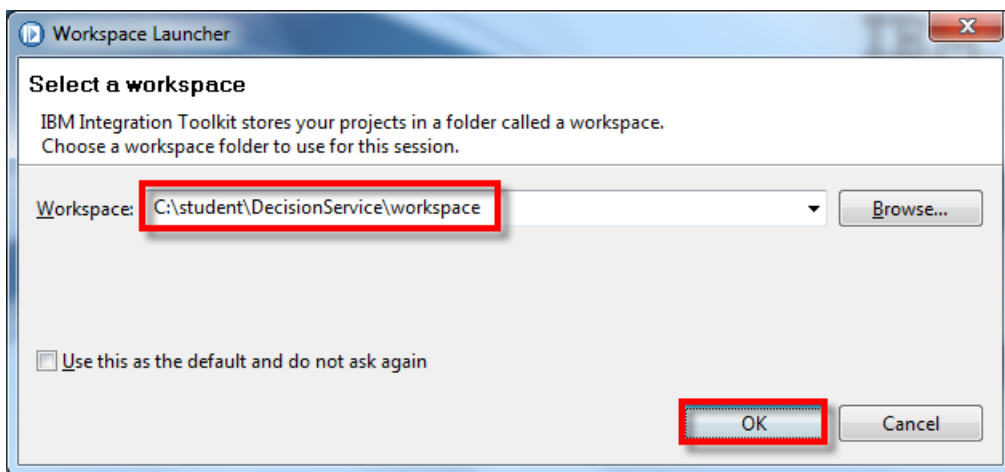
Here is an example of an action statement:

then

change the customer's category to Platinum

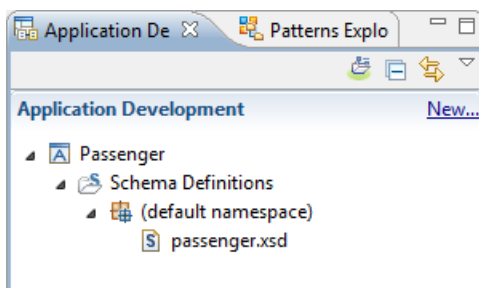
4.3 Building the business rules

1. If not already open, start the IBM Integration Bus Toolkit by double-clicking its desktop icon. Enter **c:\Student\DecisionService\workspace** as the workspace and select **OK**.

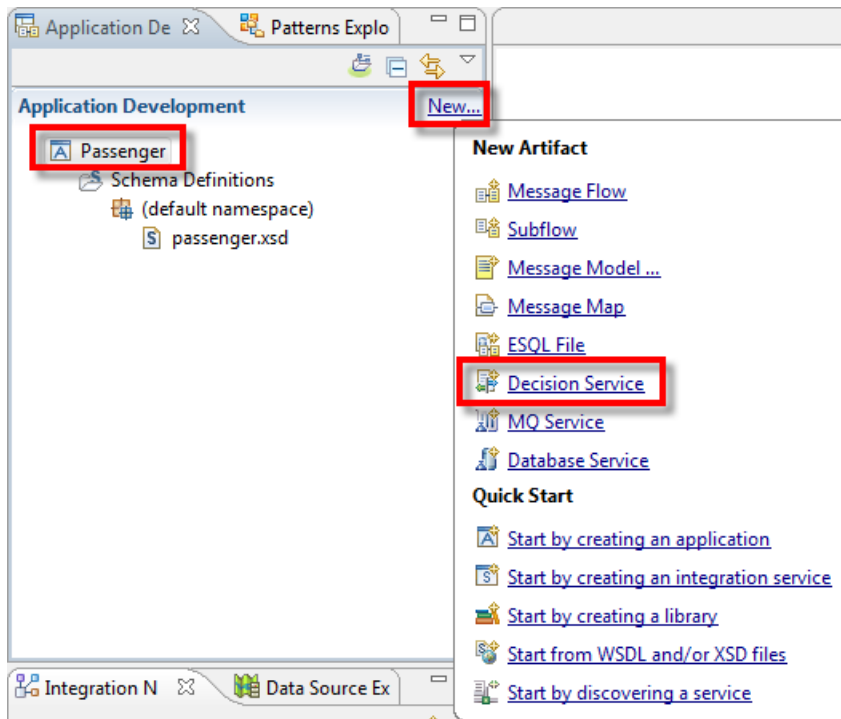


If already open, select **File→Switch Workspace→Other...** to get the workspace selection dialog, and then enter (or use Browse) **c:\Student\DecisionService\workspace** as the workspace and select **OK**.

2. The workspace has an Application named **Passenger** already created, which has an XML Schema file already loaded.



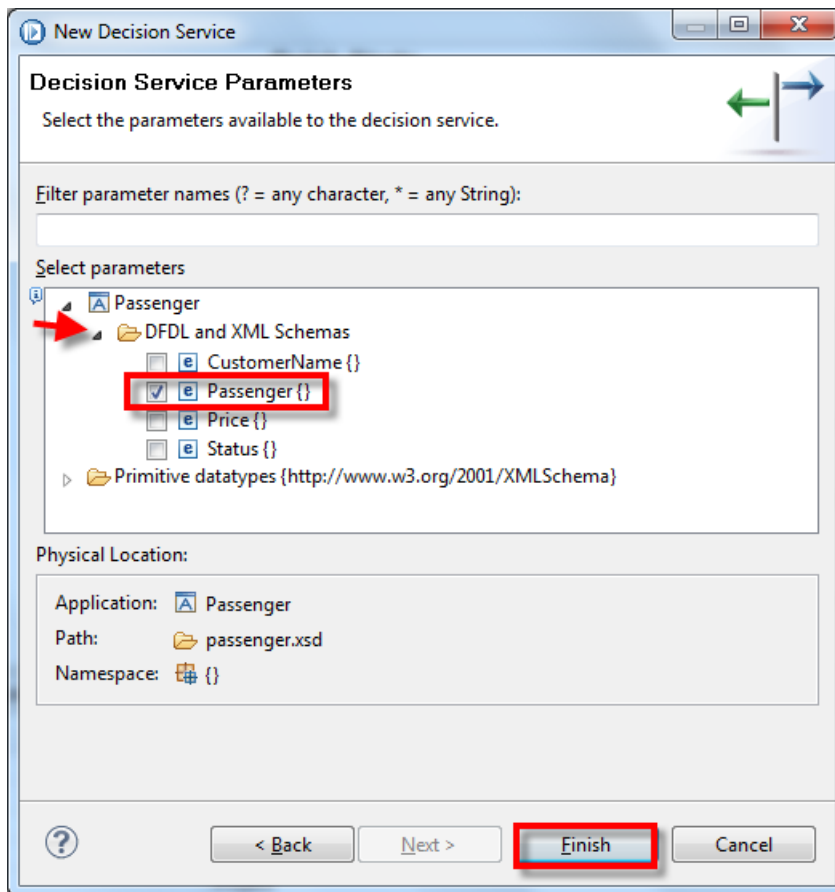
- ___3. Click on the **Passenger** Application to highlight it. Then click the **New...** link in the Application Development view, and then click **Decision Service** to create the business rules.



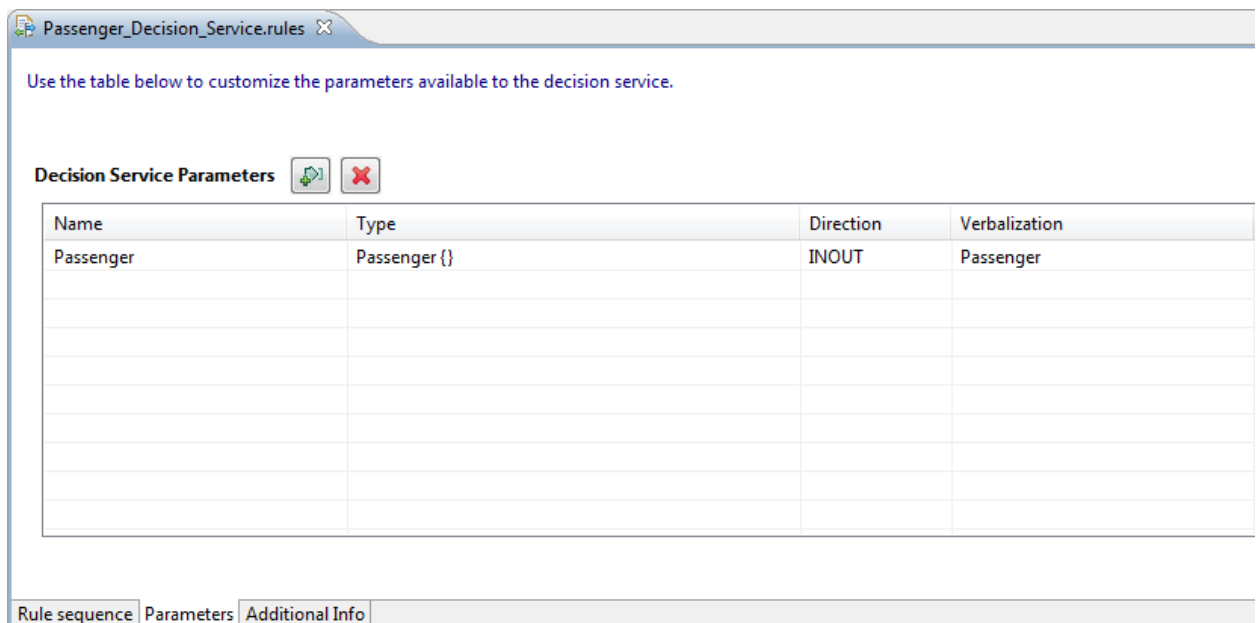
- ___4. Ensure the **Container** is **Passenger**. Enter **Passenger_Decision_Service** as the **Decision service name** and click **Next**.

The screenshot shows a window titled "New Decision Service" with a subtitle "Create a decision service" and a description "A decision service is a collection of one or more business rules". The "Container:" dropdown menu is set to "Passenger". The "Decision service name:" text box contains "Passenger_Decision_Service". At the bottom, the "Next >" button is highlighted with a red box, indicating the next step in the process.

- ___5. Click on **Passenger**, then **DFDL and XML Schemas** to open the folder of known Schemas. Click the box for **Passenger {}**. Click **Finish**.



- ___6. The Rules Editor opens up with the **Parameters** tab selected.



__7. Click on the **Rule sequence** tab.

Passenger_Decision_Service.rules

Use the table below to customize the parameters available to the decision service.

Decision Service Parameters

Name	Type	Direction	Verbalization
Passenger	Passenger {}	INOUT	Passenger

Rule sequence

Parameters

Additional Info

__8. Double-click on the **Passenger_Decision_Service** tab to make the editor full screen.

Passenger_Decision_Service.rules

Author one or more rules that will make up your decision service.
Press CTRL + SPACE to use content assist.

'Rule 1' is empty.

Rule 1

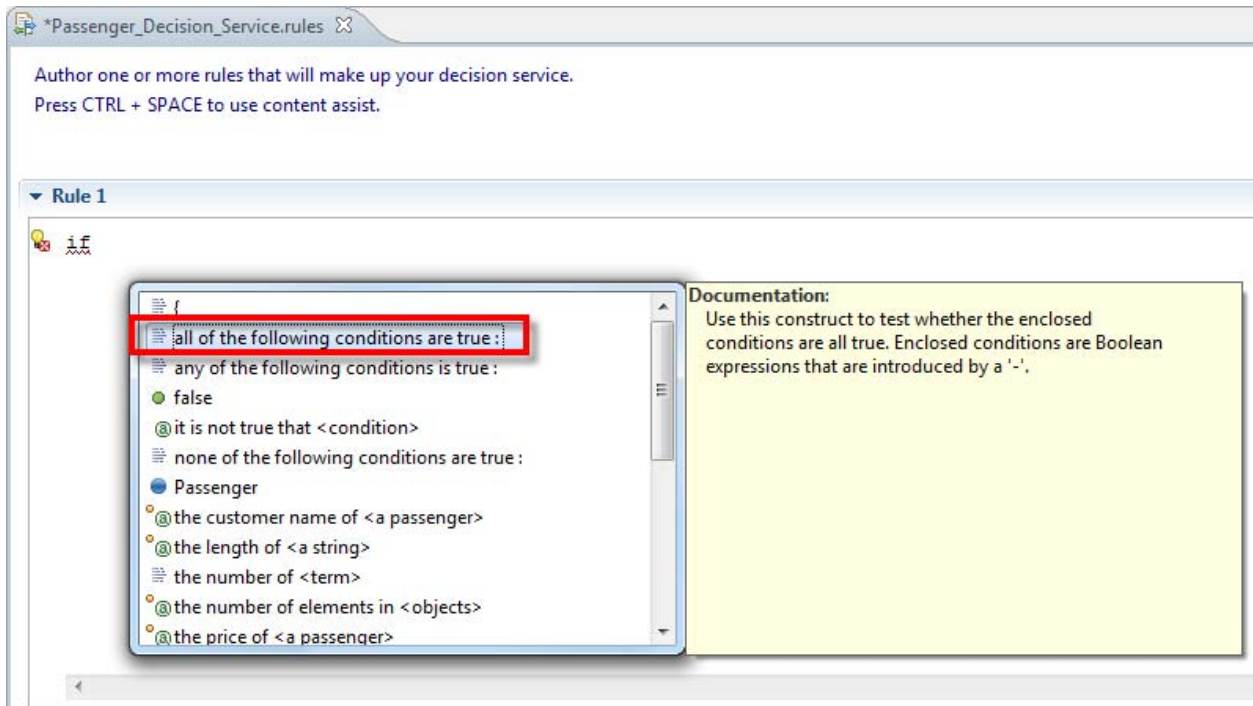
Add Rule

Rule sequence

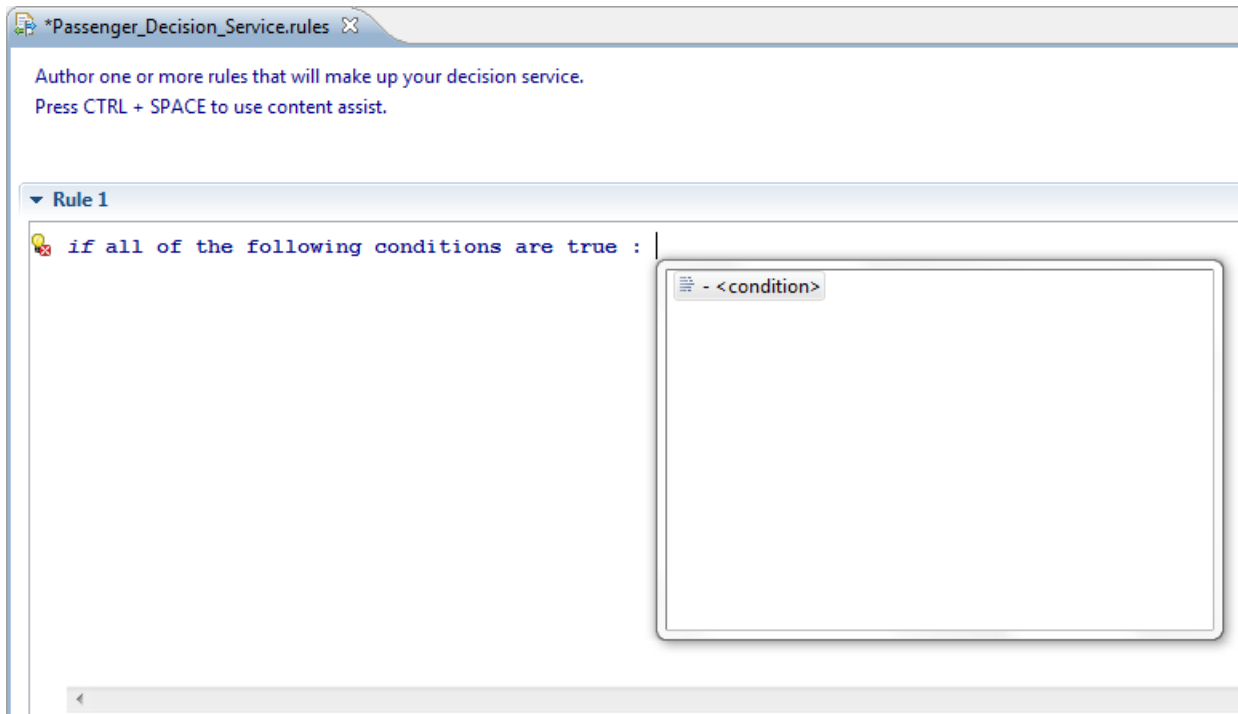
Parameters

Additional Info

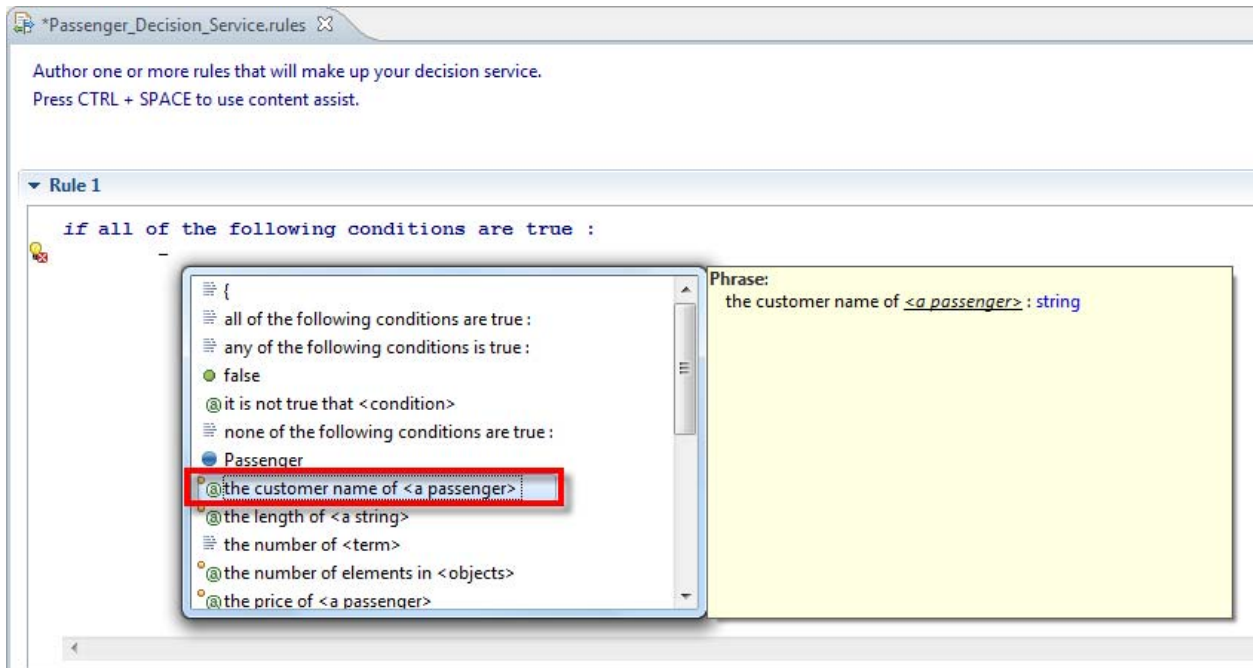
- __9. Click in the **Rule 1** window. Type **if** (if followed by a space). Content assist will pop up.



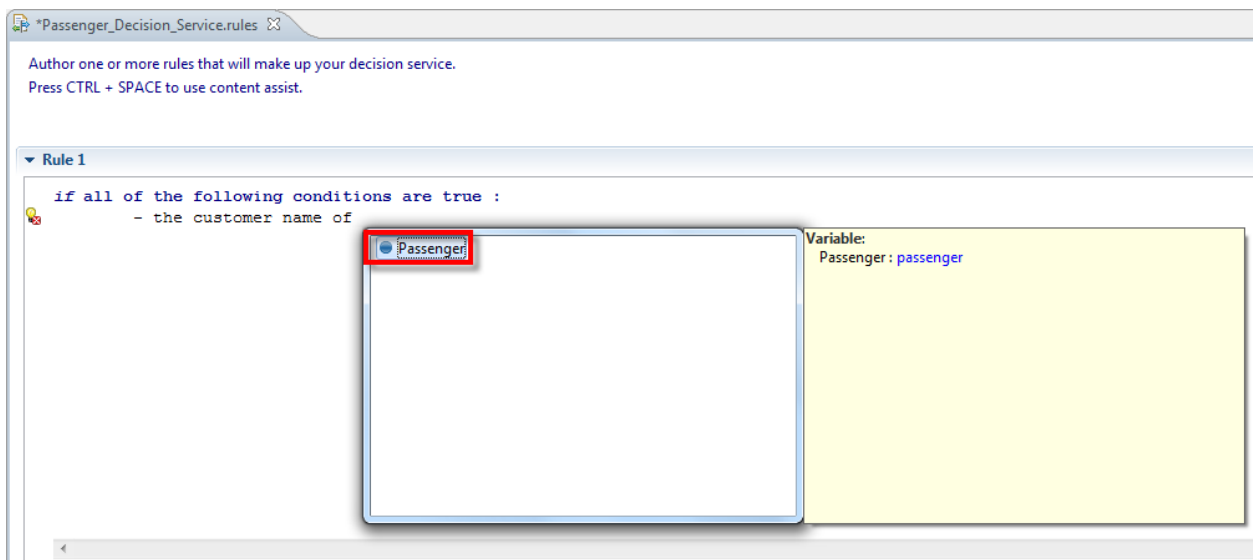
- __10. Now double-click on **all of the following conditions are true:**. Press **Enter**.



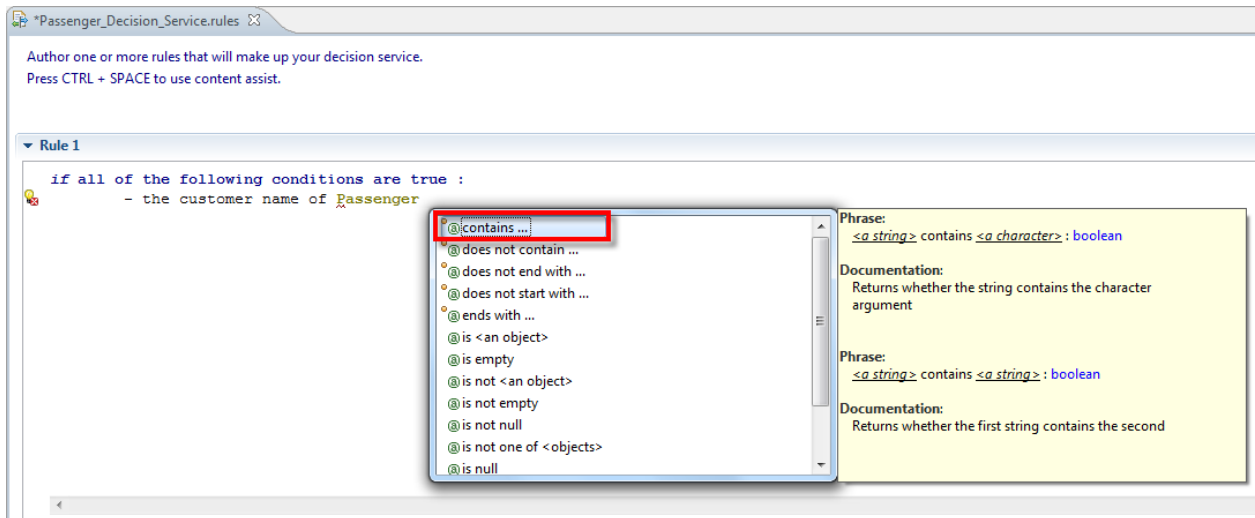
__11. Double-click on **the customer name of <a passenger>**.



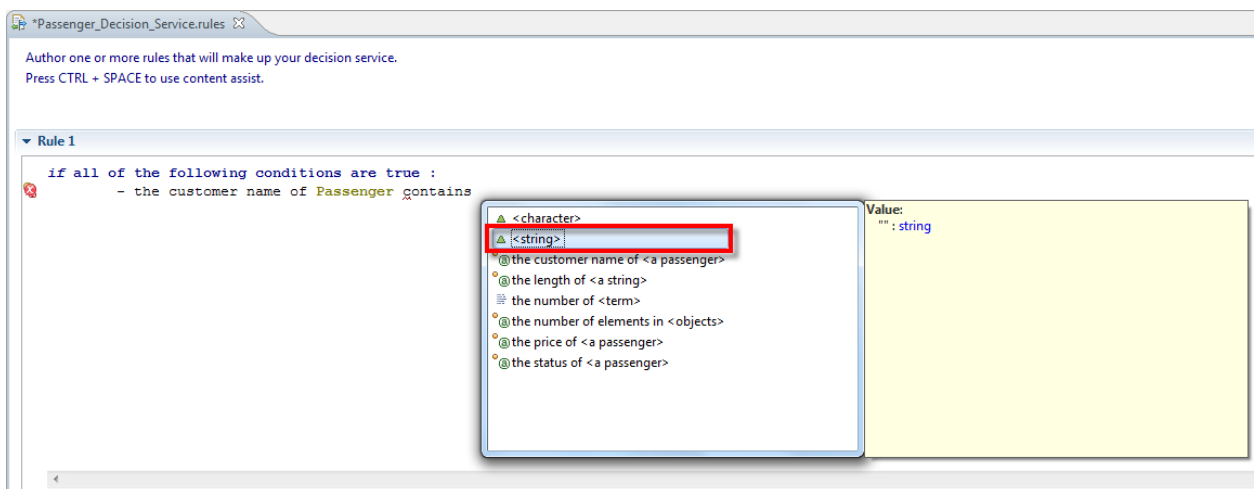
__12. Double-click **Passenger**.



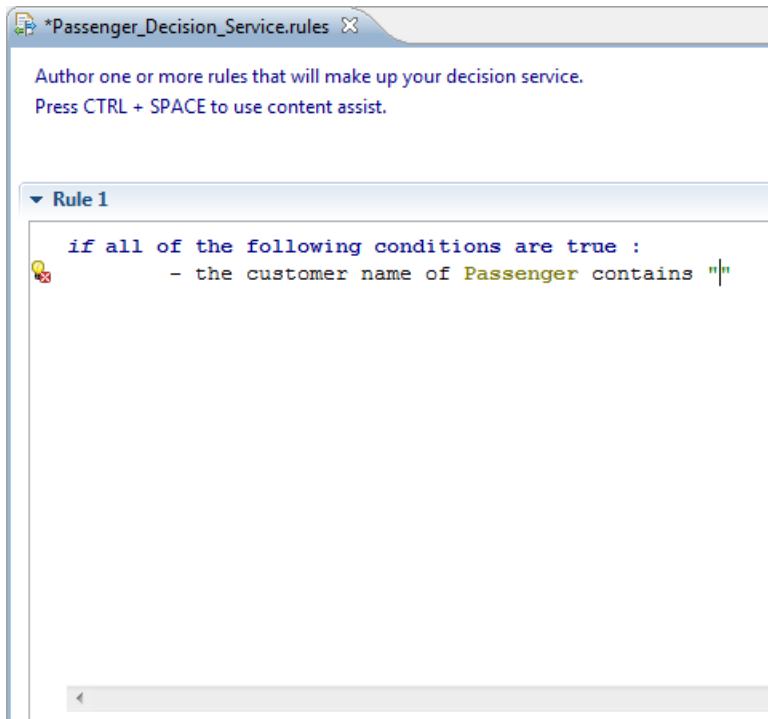
__13. Double-click on **contains...**



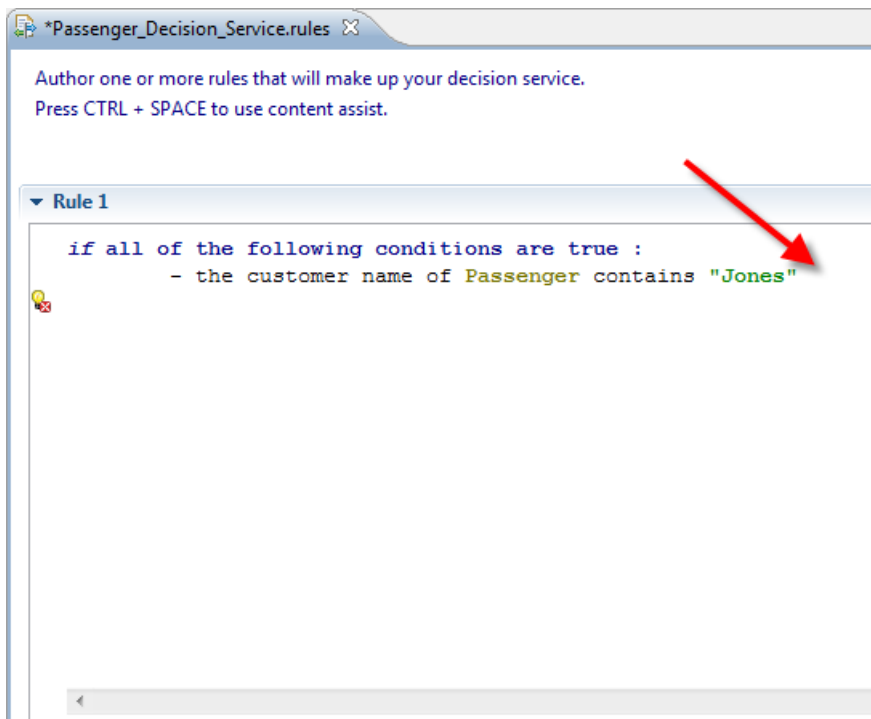
__14. Double-click <string>.



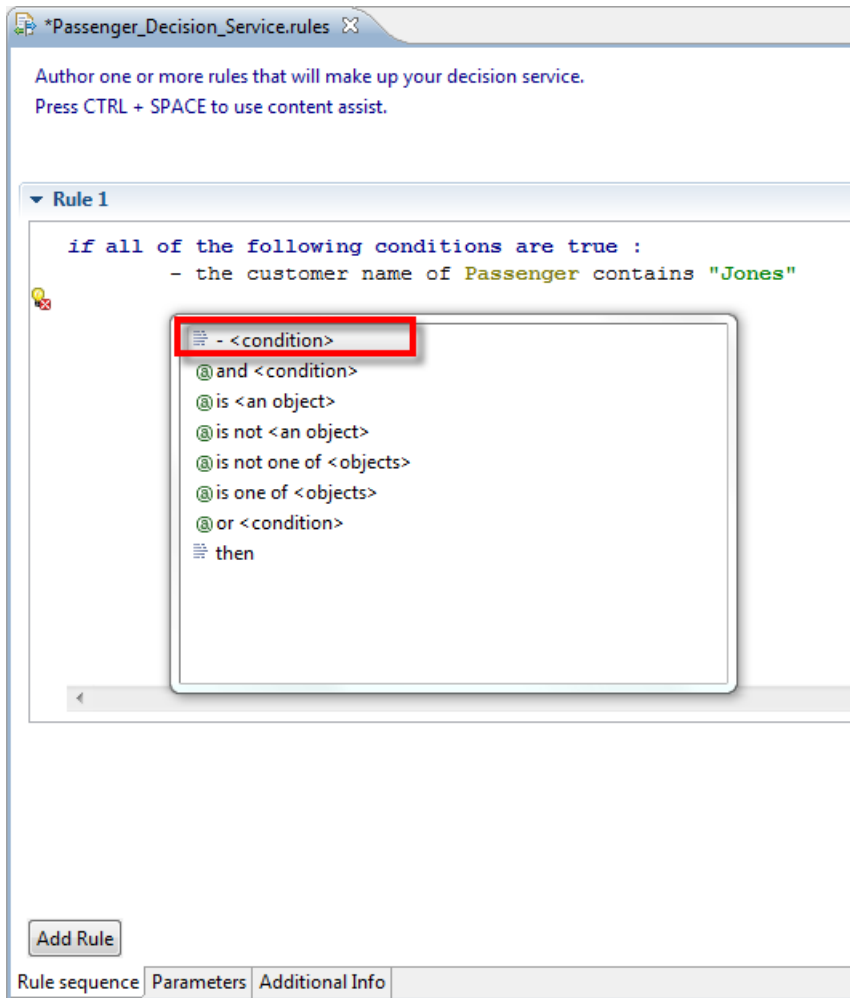
- ___15. The editor creates a set of double quotation marks. Type **Jones** between the quotation marks.



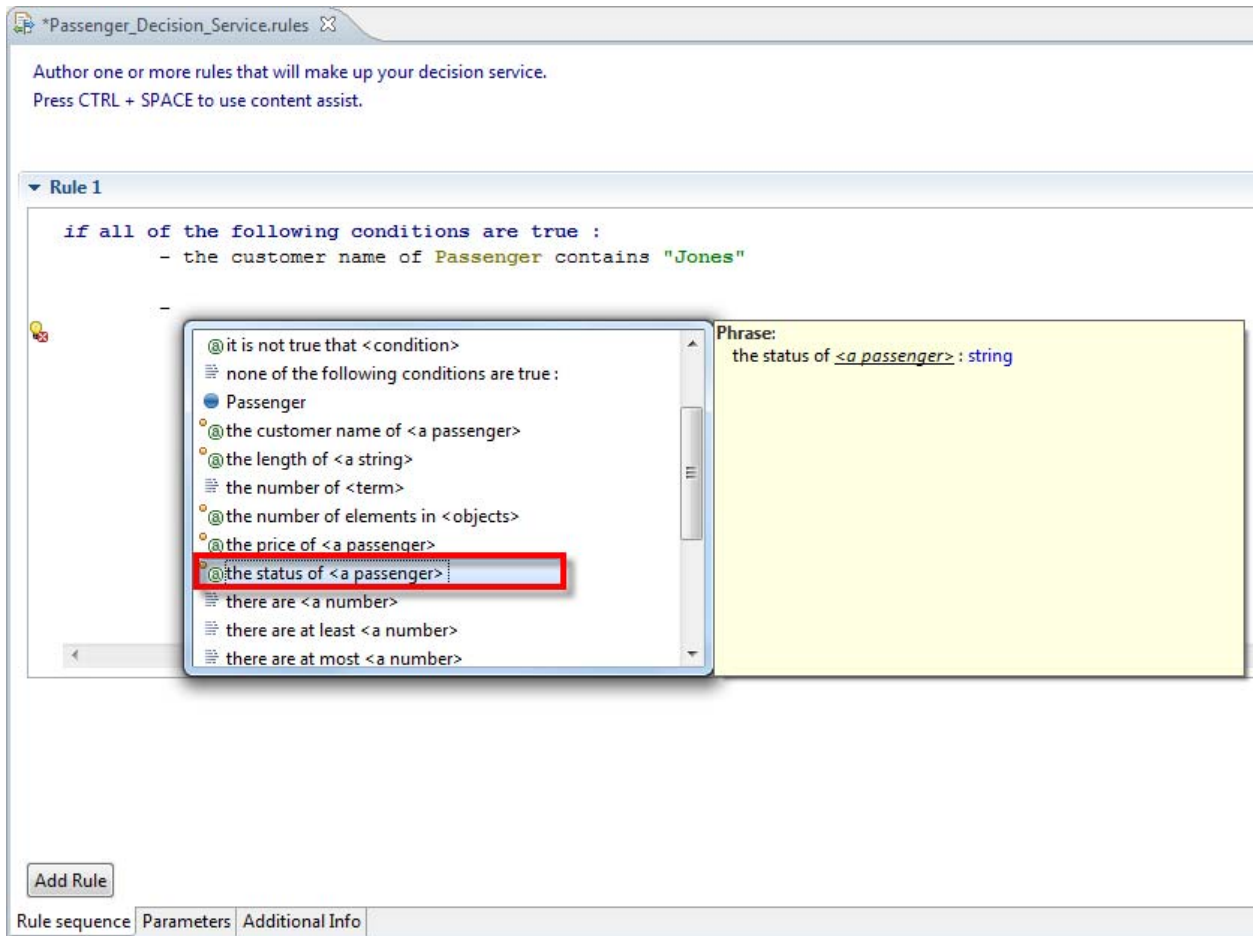
- ___16. Move the cursor past the end quotation mark and press **Enter**. You should be on a new line.



__17. Press **CTRL-Space** to invoke Content Assist again, and double-click on **<condition>**.



__18. Double-click on **the status of <a passenger>**.



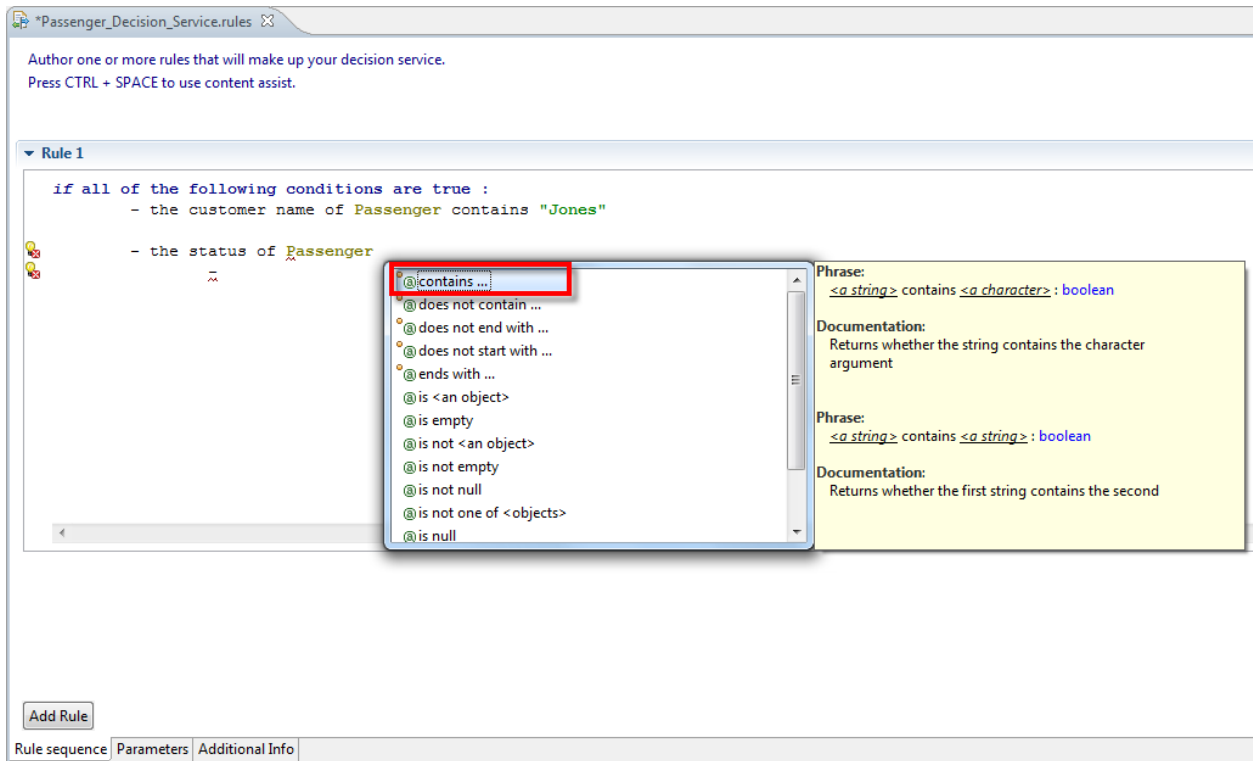
__19. Double-click on **Passenger**.

The screenshot shows the IBM Decision Service Rule Editor interface. At the top, a tab is labeled "Passenger_Decision_Service.rules". Below the tab, instructions state: "Author one or more rules that will make up your decision service. Press CTRL + SPACE to use content assist." A section titled "Rule 1" contains the following rule definition:

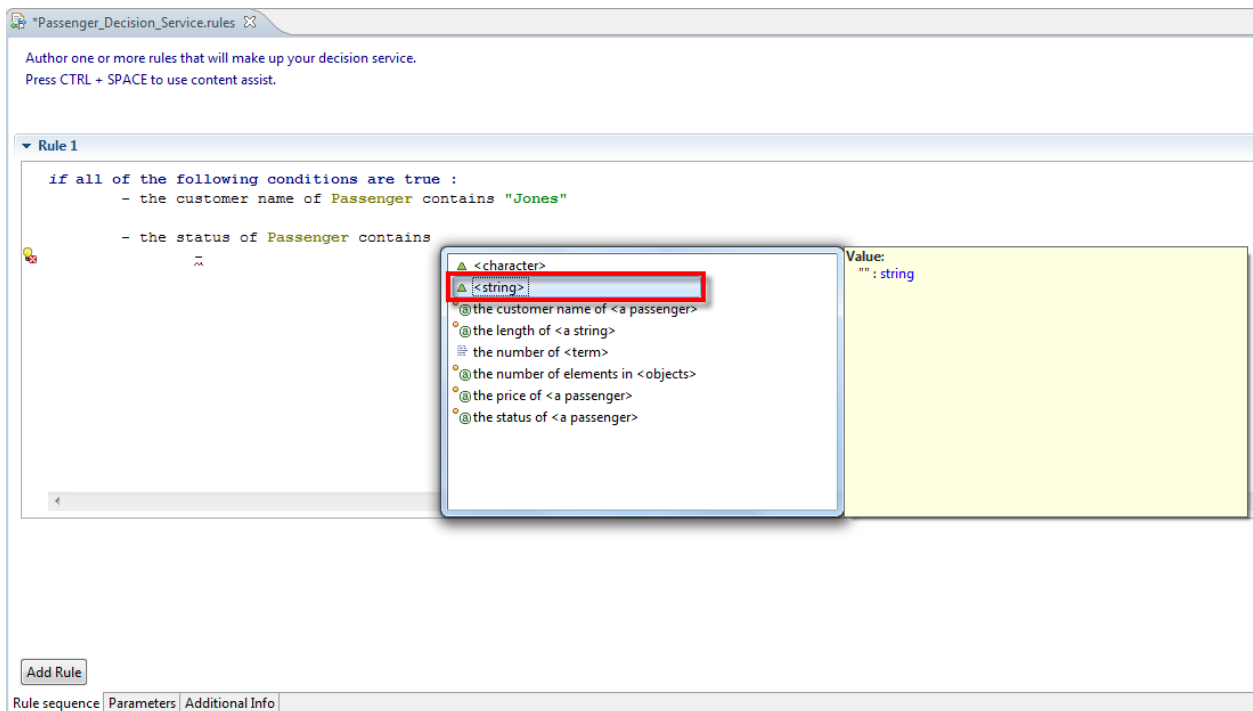
```
if all of the following conditions are true :  
  - the customer name of Passenger contains "Jones"  
  - the status of
```

A red box highlights the word "Passenger" in the first condition. To the right of the rule text, a yellow box displays the variable declaration: "Variable: Passenger : passenger". At the bottom left, there is an "Add Rule" button. At the bottom, a tabbed interface shows "Rule sequence" as the active tab, with "Parameters" and "Additional Info" tabs also visible.

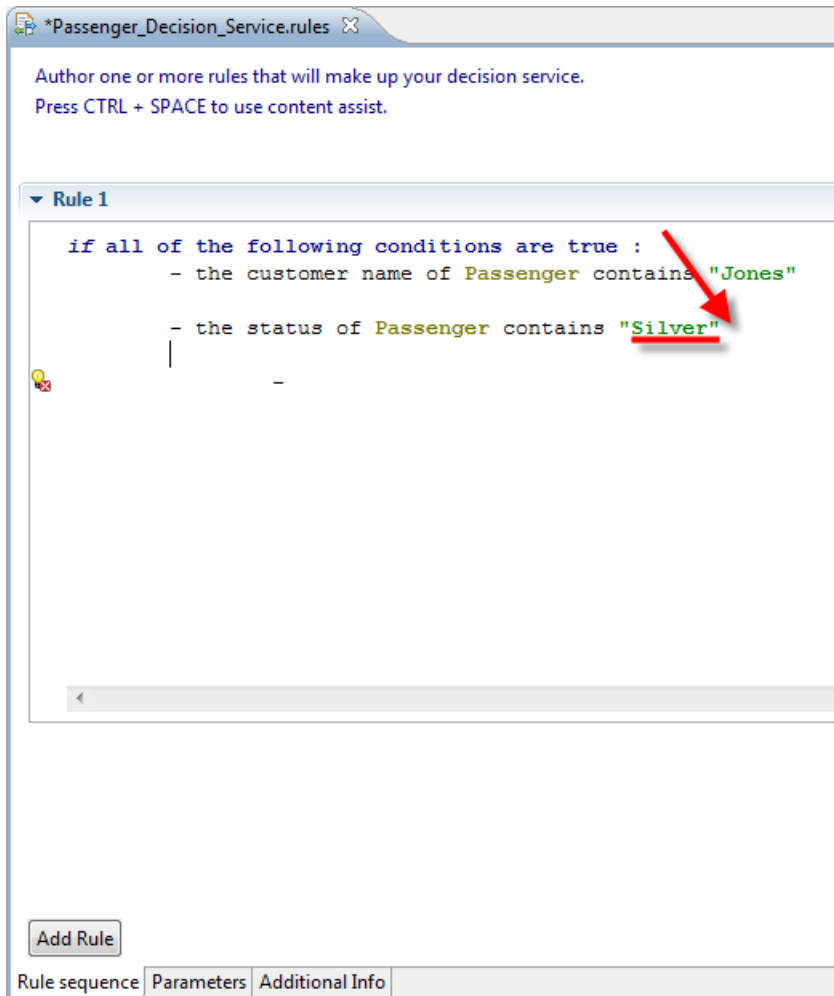
__20. Double-click on **contains...**



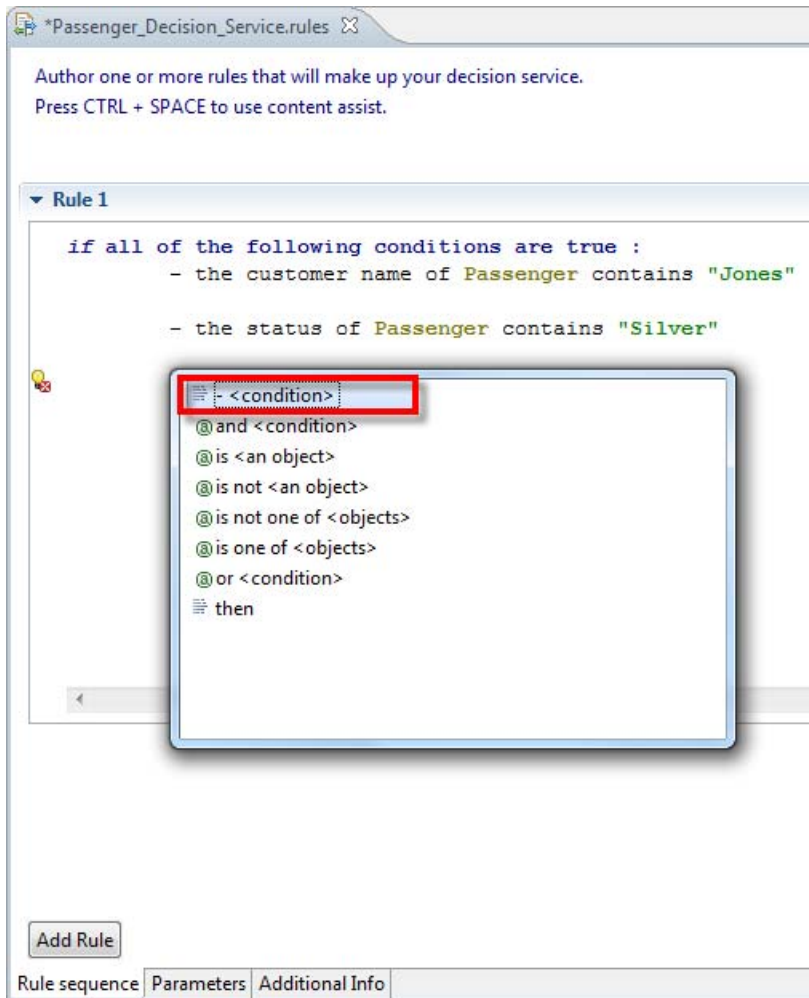
__21. Double-click on **<string>**.



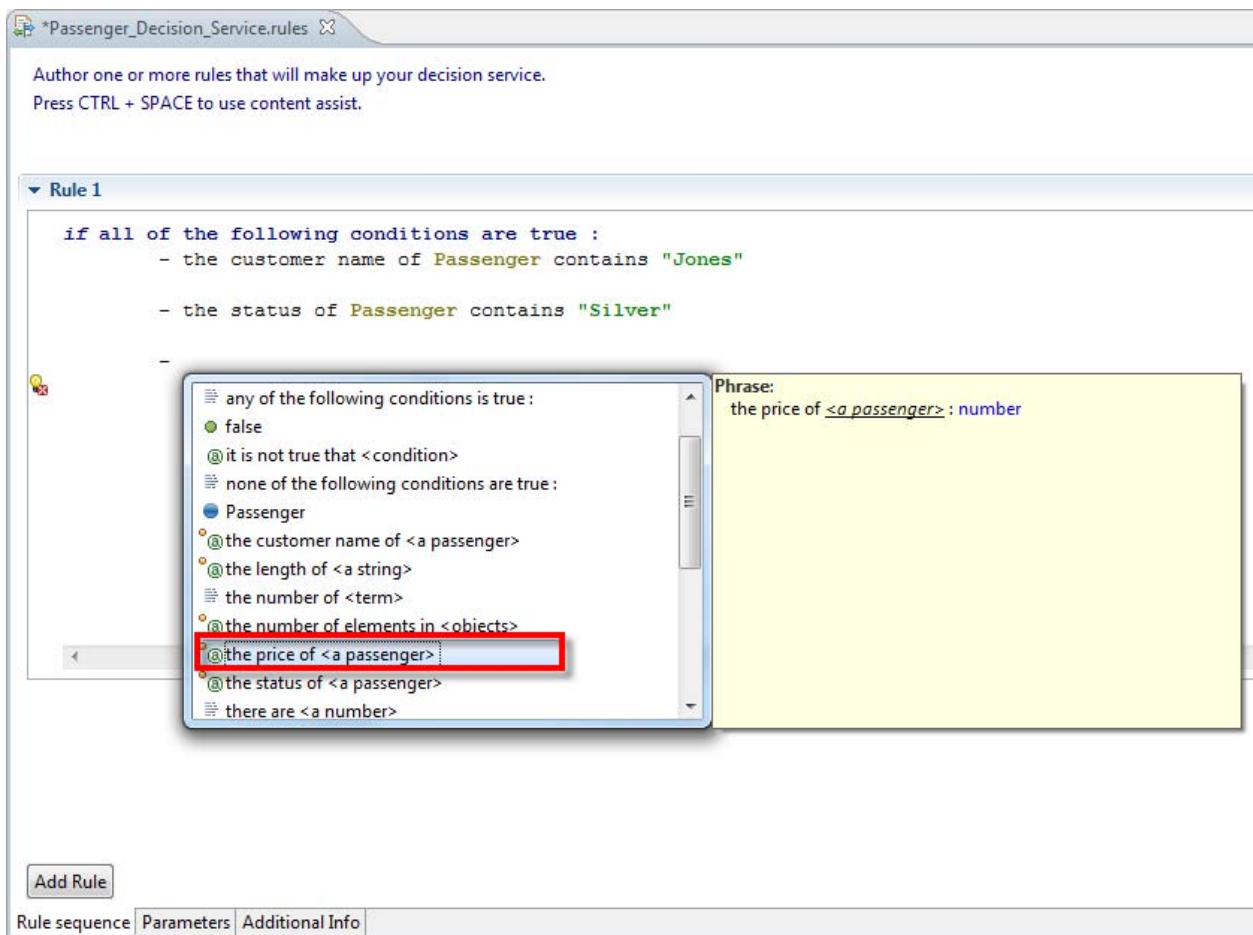
- __22. Type **Silver** between the double-quote marks, and then move the cursor past the end quotation mark and press **Enter**.



__23. Again press **CTRL-Space** to invoke Content Assist, and double-click on **<condition>**.



__24. Double-click **the price of <a passenger>**.



__25. Double-click on **Passenger**.

Author one or more rules that will make up your decision service.
Press CTRL + SPACE to use content assist.

▼ Rule 1

```
if all of the following conditions are true :  
- the customer name of Passenger contains "Jones"  
- the status of Passenger contains "Silver"  
- the price of
```

Passenger

Variable:
Passenger : passenger

Add Rule

Rule sequence | Parameters | Additional Info

__26. Double-click is at least...

Author one or more rules that will make up your decision service.
Press CTRL + SPACE to use content assist.

▼ Rule 1

```
if all of the following conditions are true :
  - the customer name of Passenger contains "Jones"
  - the status of Passenger contains "Silver"
  - the price of Passenger
```

Dropdown menu options:

- @ does not equal <a number>
- @ equals <a number>
- @ is <an object>
- @ is at least ...**
- @ is at most <a number>
- @ is between <min>
- @ is less than <a number>
- @ is more than ...
- @ is not <an object>
- @ is not one of <objects>
- @ is one of <objects>
- @ is strictly between <min>

Phrase: <a number> is at least <a number> : boolean

Documentation: Returns whether the first numeric value is greater than or equal to the second

Phrase: <a number> is at least <min> and less than <max> : boolean

Documentation: Returns whether the first numeric value is greater than or equal to the second and smaller than the third

Add Rule

Rule sequence | Parameters | Additional Info

__27. Double-click on <number>.

Author one or more rules that will make up your decision service.
Press CTRL + SPACE to use content assist.

▼ Rule 1

```
if all of the following conditions are true :
  - the customer name of Passenger contains "Jones"
  - the status of Passenger contains "Silver"
  - the price of Passenger is at least
```

Dropdown menu options:

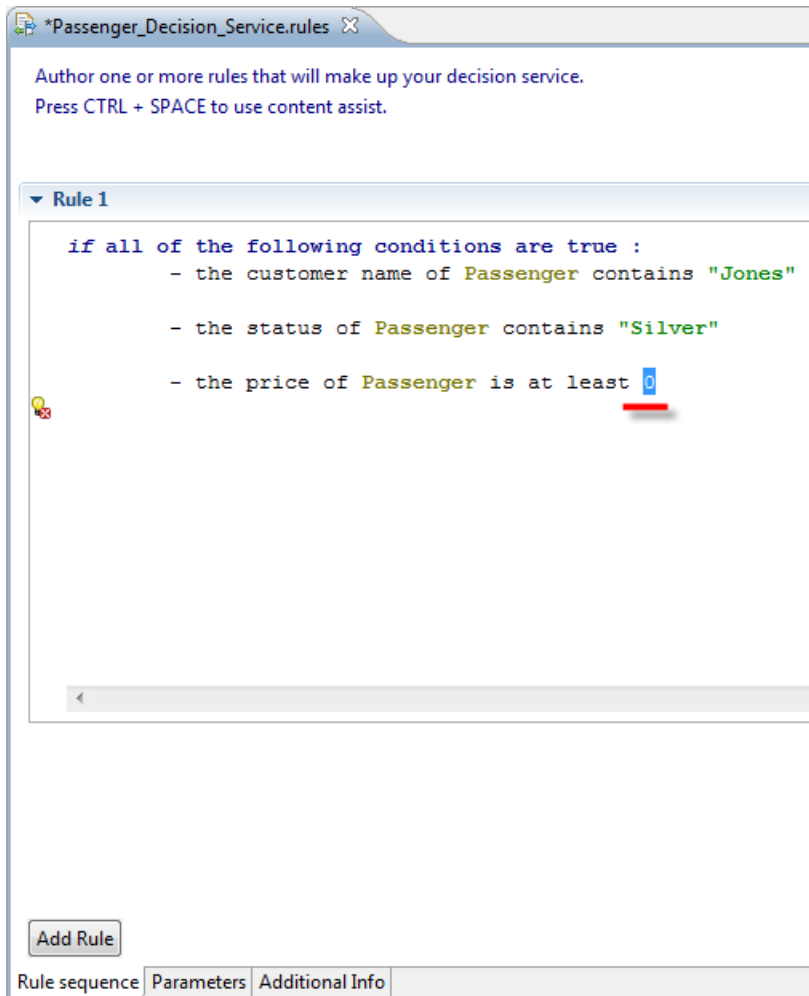
- <number>**
- @ the length of <a string>
- @ the number of <term>
- @ the number of elements in <objects>
- @ the price of <a passenger>

Value: 0 : number

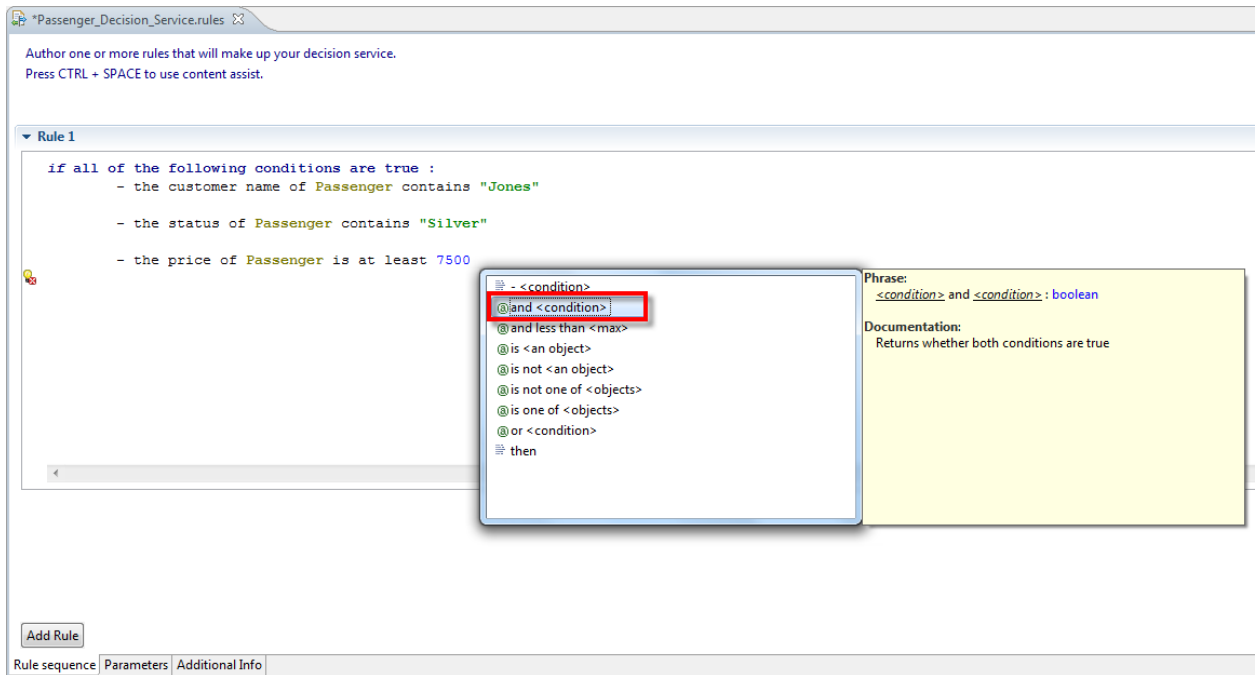
Add Rule

Rule sequence | Parameters | Additional Info

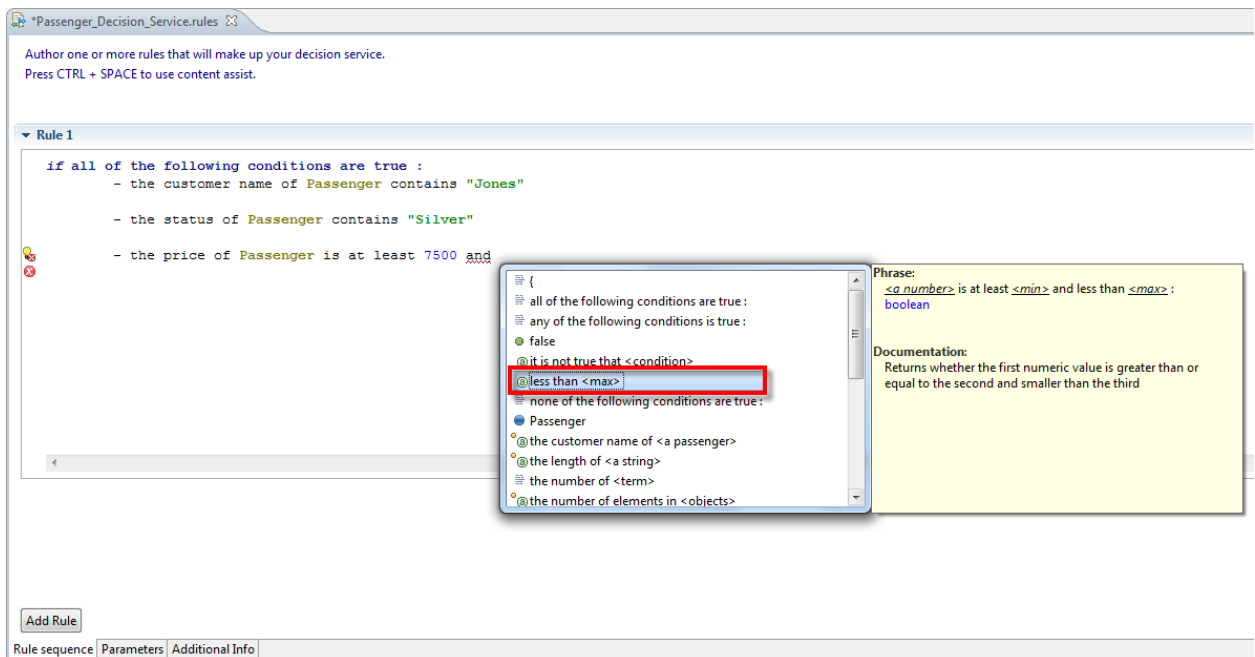
- ___28. A zero (0) will be inserted, and the cursor will be placed over it. Type **7500** and a **space** as the new value, replacing the 0.



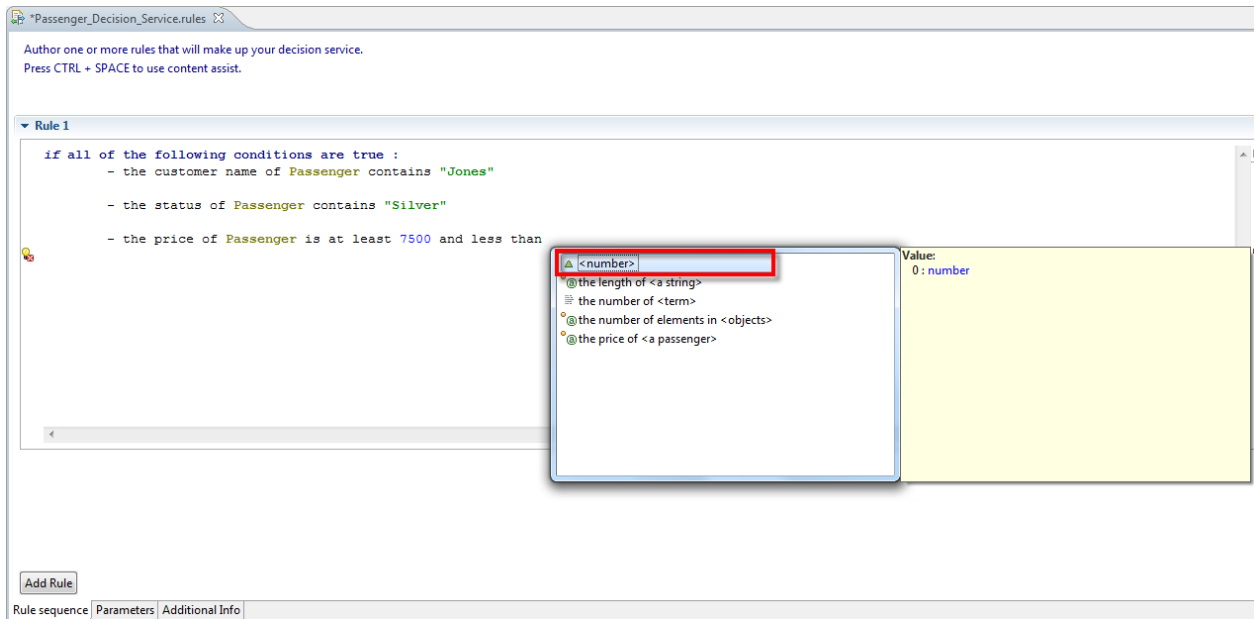
- __29. The editor should look as follows at this point, with content assist popped up due to typing the space. Double-click **and** **<condition>**.



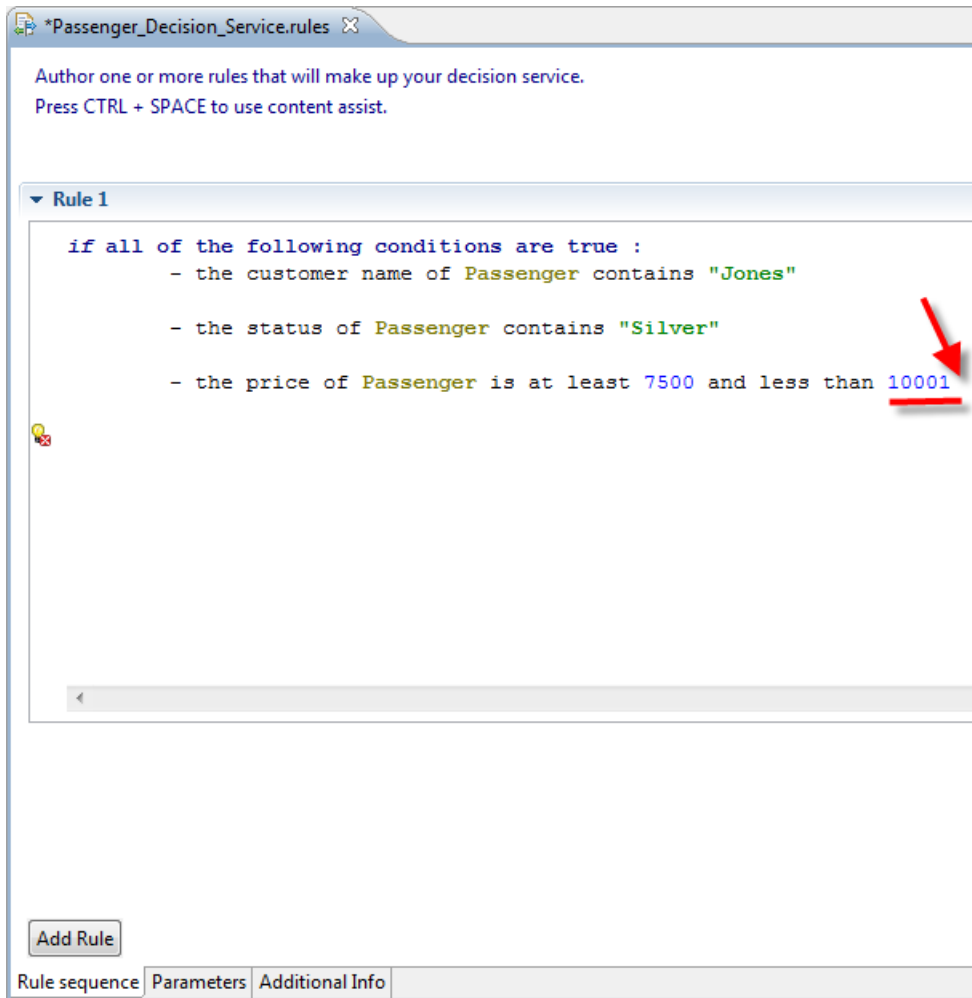
- __30. Double-click **less than** **<max>**.



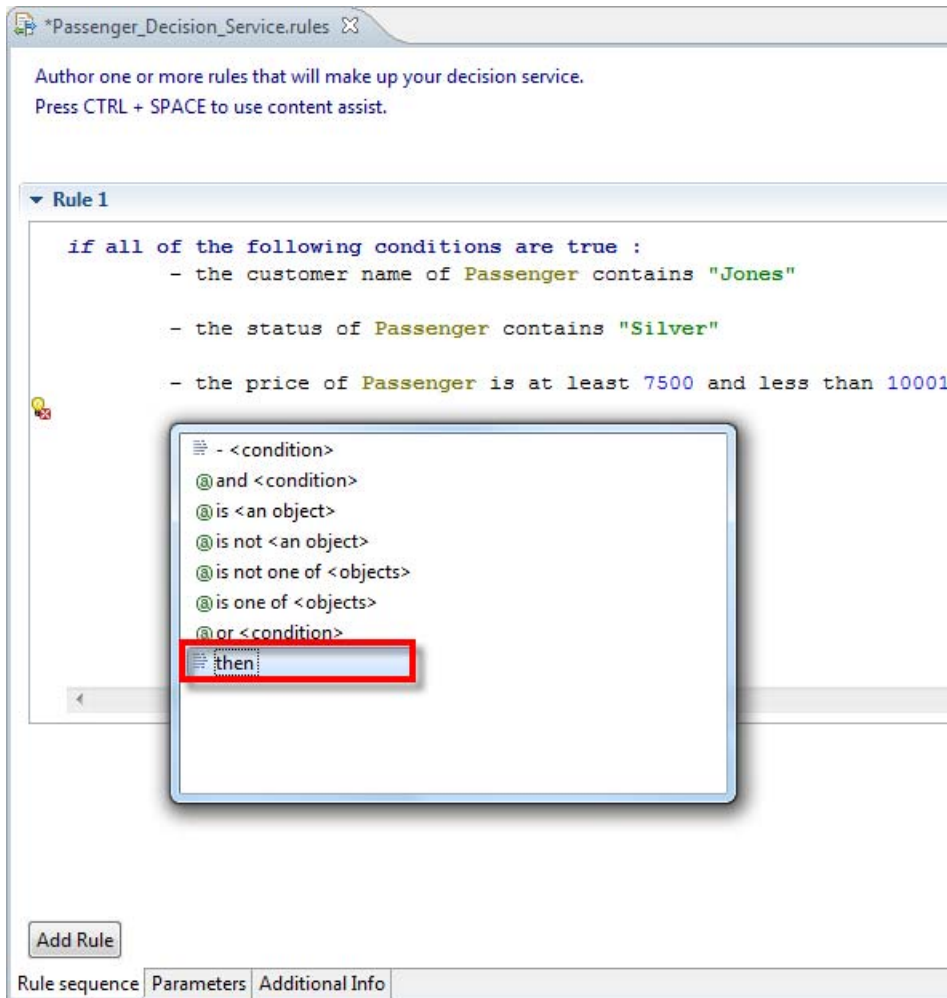
__31. Double-click on **<number>**.



__32. Type **10001** and press **Enter**.



__33. Press **CTRL-Space** to invoke Content Assist again, and double-click on **then**.



__34. Double-click **set the status of <a passenger>**.

The screenshot shows the IBM Decision Service Rule Editor interface. At the top, a tab is labeled '*Passenger_Decision_Service.rules'. Below the tab, instructions state: 'Author one or more rules that will make up your decision service. Press CTRL + SPACE to use content assist.'

Under the 'Rule 1' section, the rule logic is defined as follows:

```
if all of the following conditions are true :  
  - the customer name of Passenger contains "Jones"  
  - the status of Passenger contains "Silver"  
  - the price of Passenger is at least 7500 and less than 10001  
  
then  
  for each <term>  
    @print <a string>  
    set <variable>  
    @set the customer name of <a passenger>  
    @set the price of <a passenger>  
    @set the status of <a passenger>
```

The action '@set the status of <a passenger>' is highlighted with a red rectangular box. To the right of the rule editor, a 'Phrase:' box contains the text: 'set the status of <a passenger> to <a string>'.

At the bottom left, there is an 'Add Rule' button. At the bottom, a tabbed interface shows 'Rule sequence', 'Parameters', and 'Additional Info'.

__35. Double-click **Passenger**.

The screenshot shows the IBM Integration Bus Rule Editor interface. At the top, a tab is labeled "Passenger_Decision_Service.rules". Below the tab, a message states: "Author one or more rules that will make up your decision service. Press CTRL + SPACE to use content assist." A section titled "Rule 1" contains the following rule definition:

```
if all of the following conditions are true :  
  - the customer name of Passenger contains "Jones"  
  - the status of Passenger contains "Silver"  
  - the price of Passenger is at least 7500 and less than 10001  
then set the status of
```

A context menu is open over the word "Passenger" in the "then" clause, with "Passenger" highlighted. To the right of the menu, a yellow box displays "Variable: Passenger : passenger". At the bottom left, there is an "Add Rule" button. At the bottom, a tabbed interface shows "Rule sequence" as the active tab, with "Parameters" and "Additional Info" tabs also visible.

__36. Double-click to **<a string>**.

Author one or more rules that will make up your decision service.
Press CTRL + SPACE to use content assist.

▼ Rule 1

```

if all of the following conditions are true :
    - the customer name of Passenger contains "Jones"
    - the status of Passenger contains "Silver"
    - the price of Passenger is at least 7500 and less than 10001
then set the status of Passenger
  
```

@to <a string>

Phrase:
set the status of <a passenger> to <a string>

Add Rule

Rule sequence | Parameters | Additional Info

__37. Double-click **<string>**.

Author one or more rules that will make up your decision service.
Press CTRL + SPACE to use content assist.

▼ Rule 1

```

if all of the following conditions are true :
    - the customer name of Passenger contains "Jones"
    - the status of Passenger contains "Silver"
    - the price of Passenger is at least 7500 and less than 10001
then set the status of Passenger to
  
```

<string>

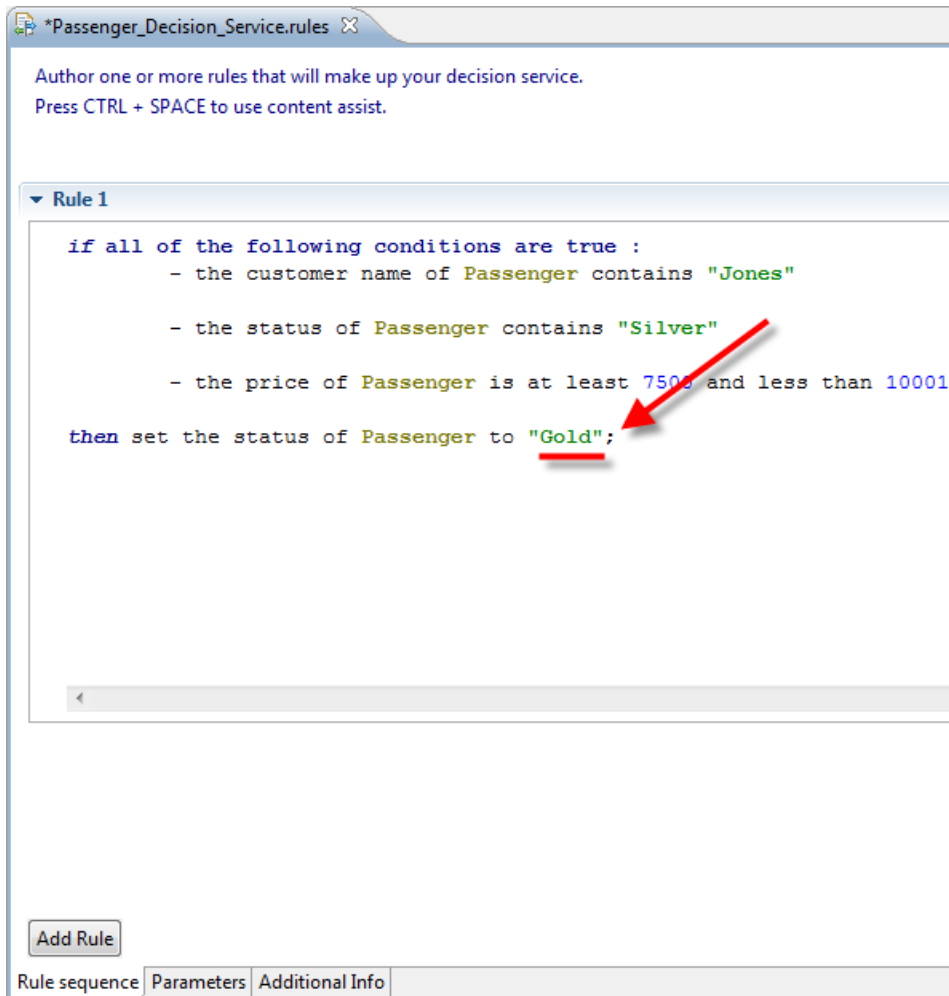
- @the customer name of <a passenger>
- @the length of <a string>
- the name of this rule
- the number of <term>
- @the number of elements in <objects>
- @the price of <a passenger>
- @the status of <a passenger>

Value:
"": string

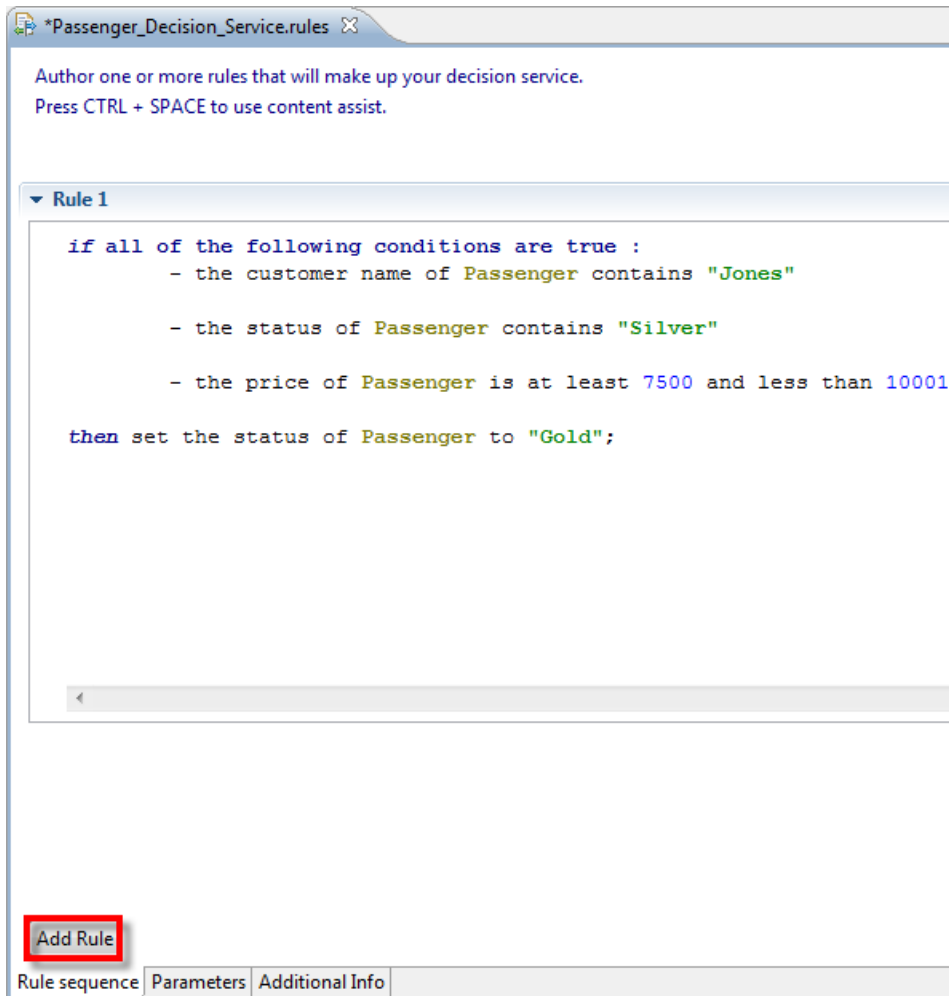
Add Rule

Rule sequence | Parameters | Additional Info

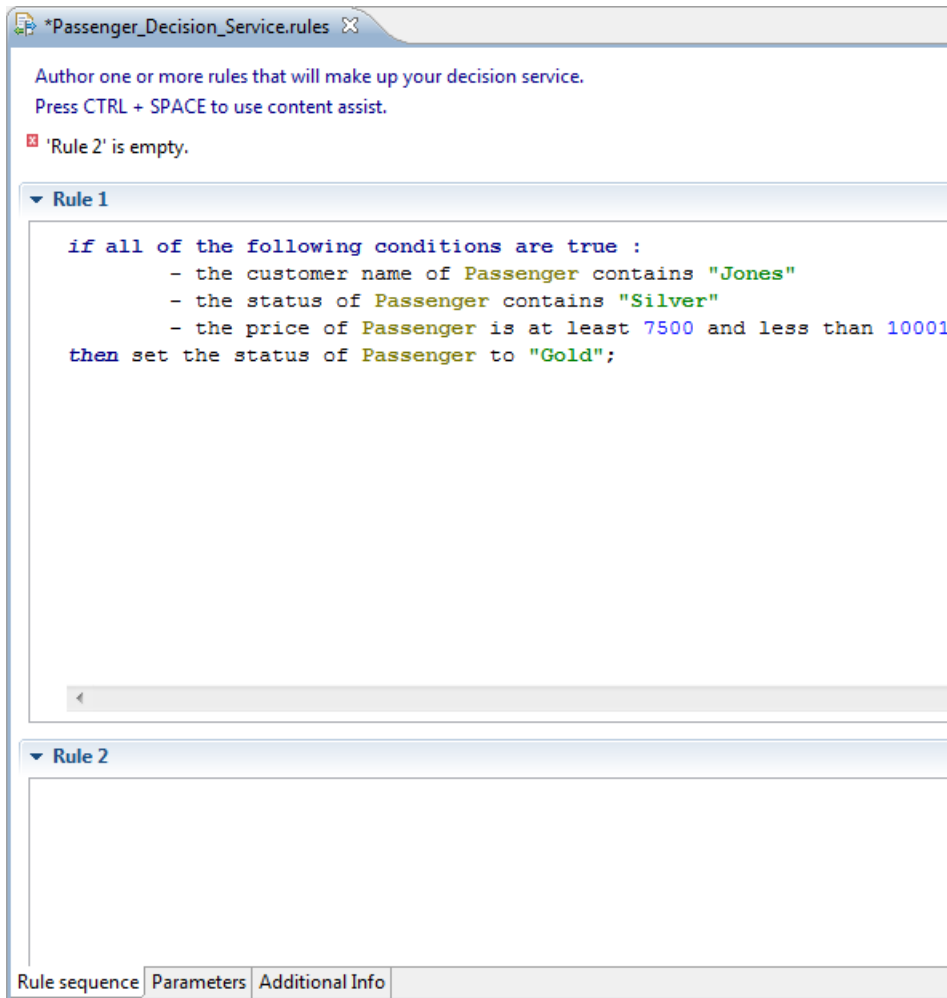
- ___38. Type **Gold** between double quotation marks. Move the cursor past the end quotation mark, and type a semi-colon (;).



- ___39. The first rule is complete. It should look as follows. Press **CTRL-S** to save the rules so far. Then click on **Add Rule** to add the second (blank) rule into the editor.



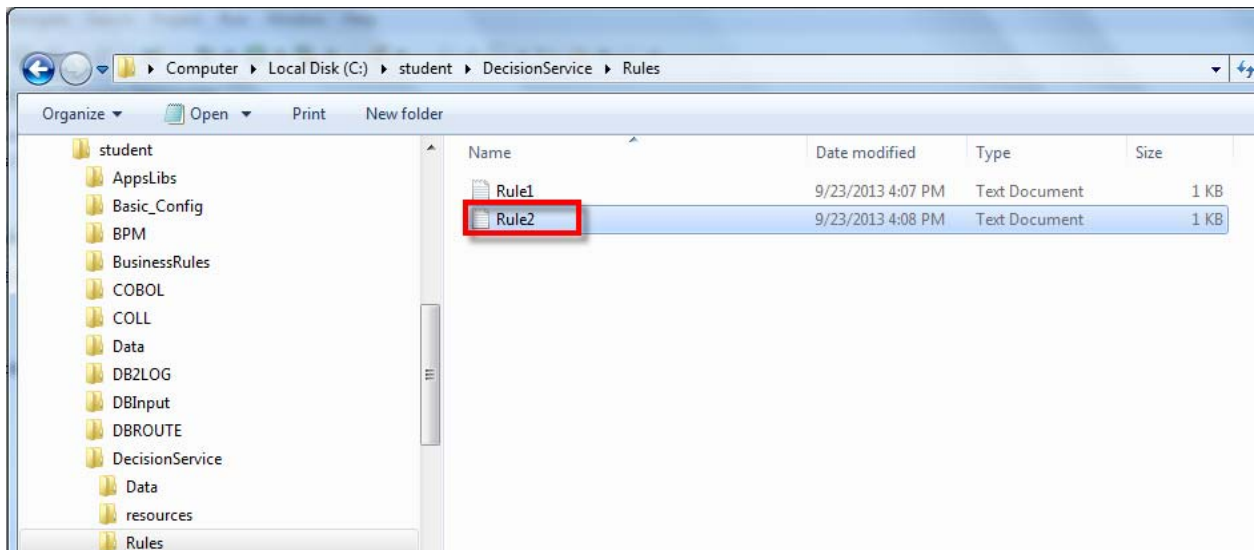
__40. The empty second rule is added to the editor.



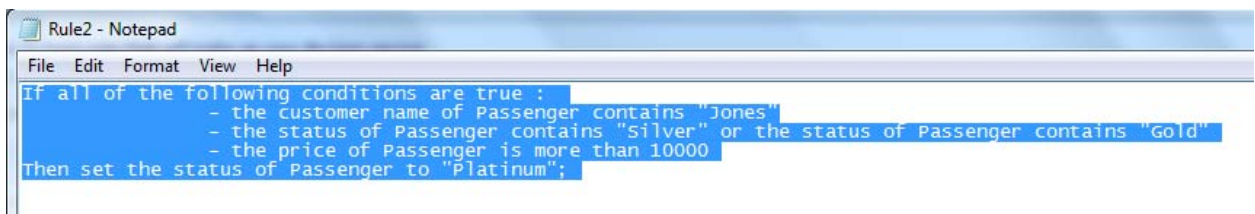
__41. For the second rule, we will paste the rule into the editor. Open a Windows Explorer session. You can click on the icon in the Task Bar.



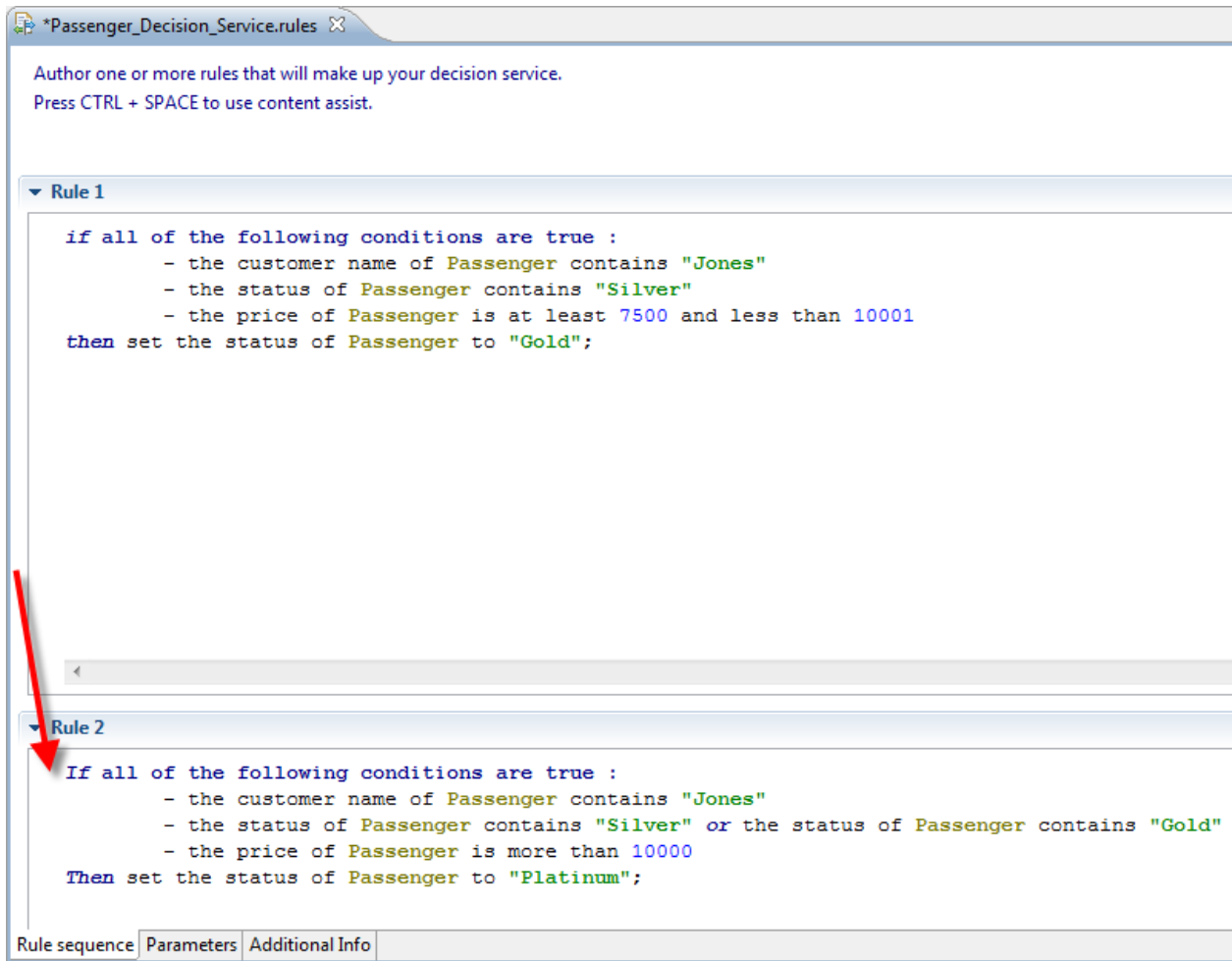
- __42. Navigate to **C:\student\DecisionService\Rules** and open the **Rule2.txt** file with Notepad (you can double-click on the file name).



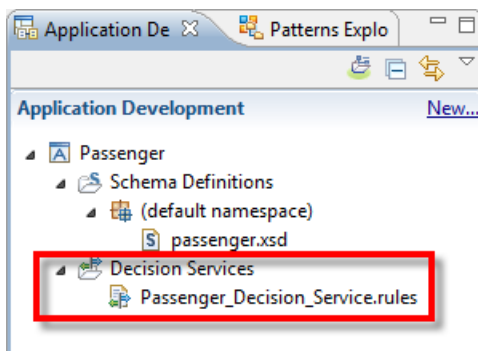
- __43. Press **CTRL-A** to select the entire text, and then press **CTRL-C** to copy the selected text.



- ___44. Return to the Integration Bus Toolkit. Position your cursor at the start of Rule 2, and then press **CTRL-V** to paste the text into the editor.



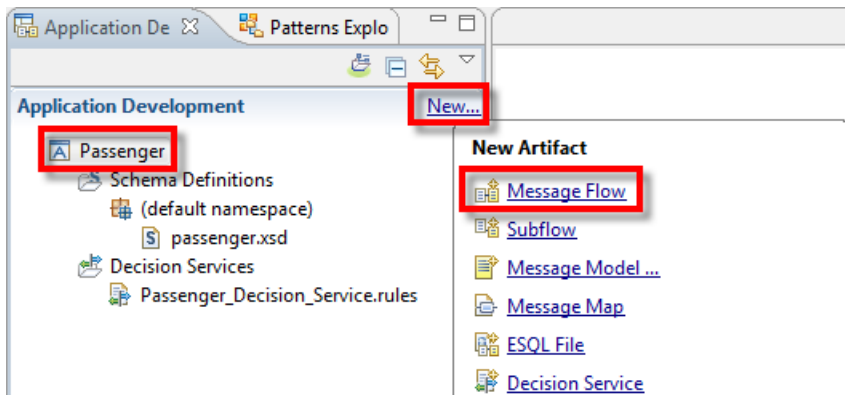
- ___45. Press **CTRL-S** to save the rules. Close the rules editor.
- ___46. In the Application Development explorer view, you will see the **Passenger_Decision_Service.rules** file you created, within a Decision Services folder.



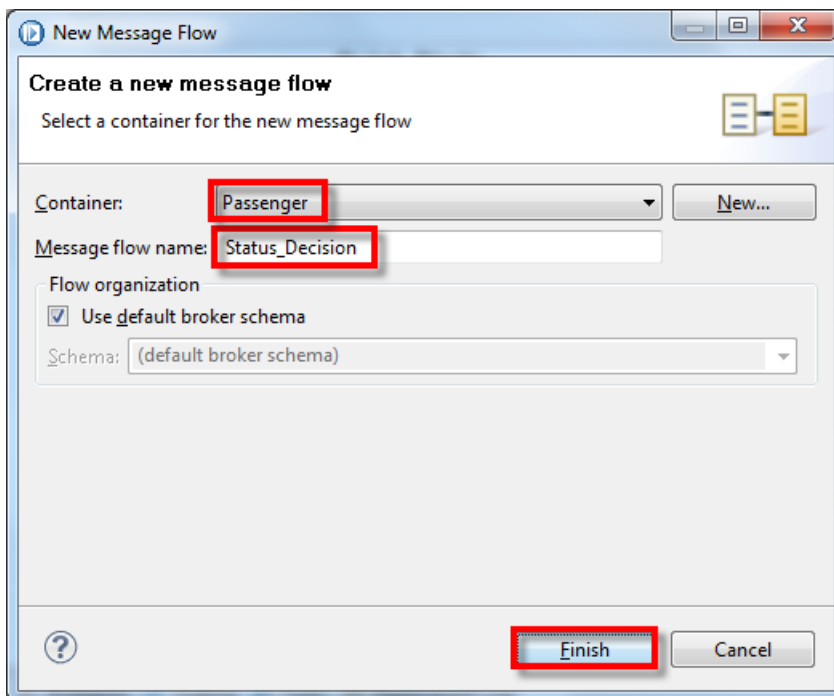
4.4 Build the message flow

You will now build the message flow that uses the rules that you just created.

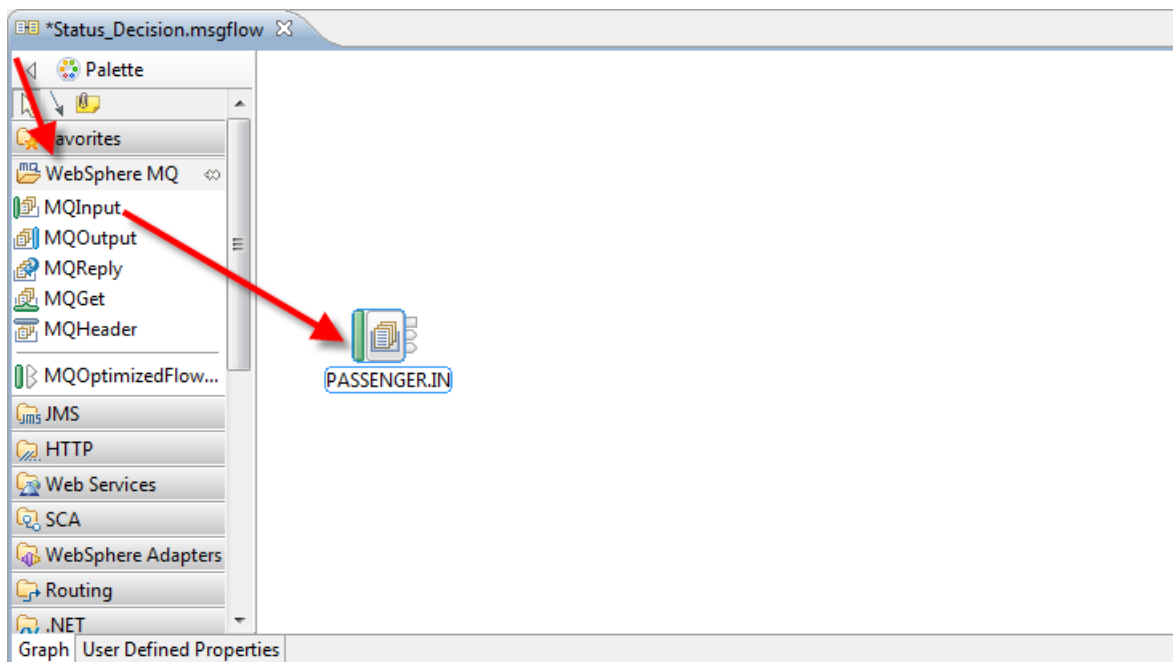
1. Click on **Passenger** in the Application Development view to highlight it, then click the **New...** quick start link. Click on **Message Flow** from the New Artifact popup.



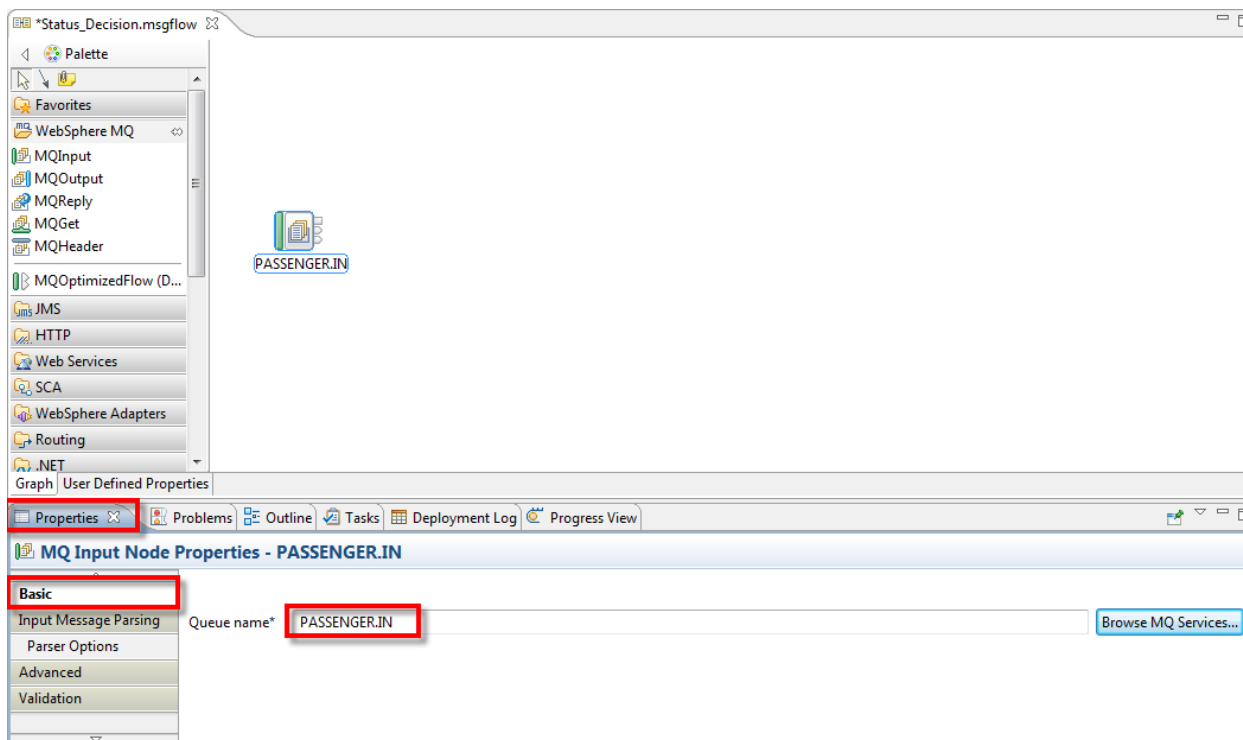
2. Ensure the **Container** is **Passenger**. Enter **Status_Decision** as the **Message flow name** and click **Finish**.



- ___3. Open the **WebSphere MQ** folder in the Palette, and drag a **MQInput** node onto the canvas. Name the node **PASSENGER.IN**.

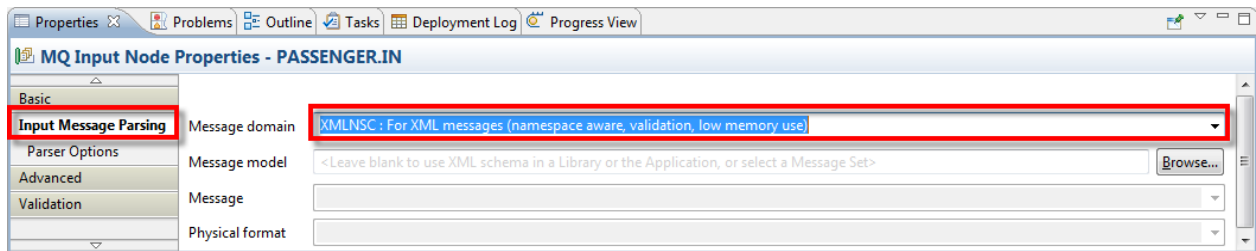


- ___4. In the **Properties** for the MQInput node, on **Basic** tab, enter **PASSENGER.IN** as the Queue Name.

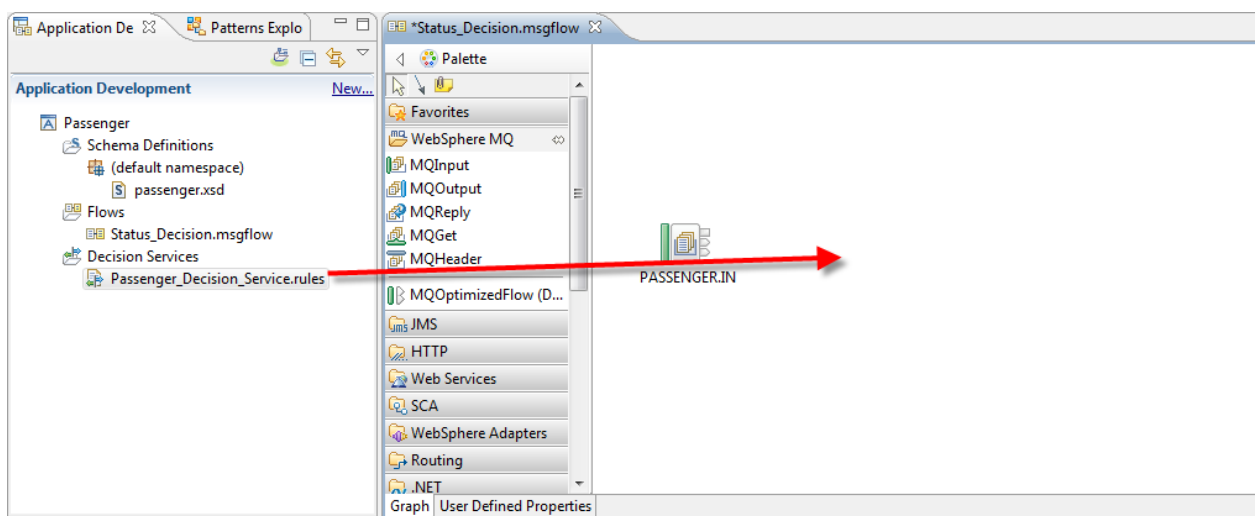


Note: Queue names are case sensitive.

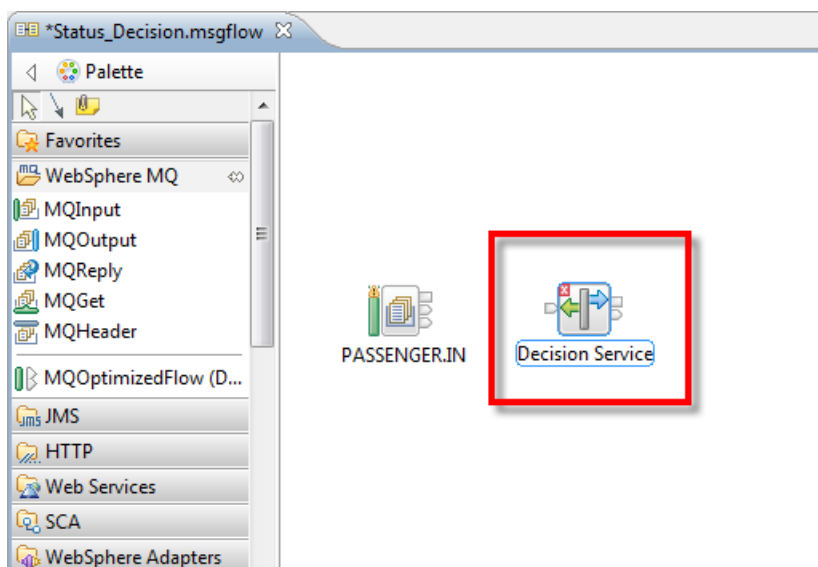
- __5. Still in Properties, select the **Input Message Parsing** tab. Use the drop down on the Message domain box, and select **XMLNSC** as the parser.



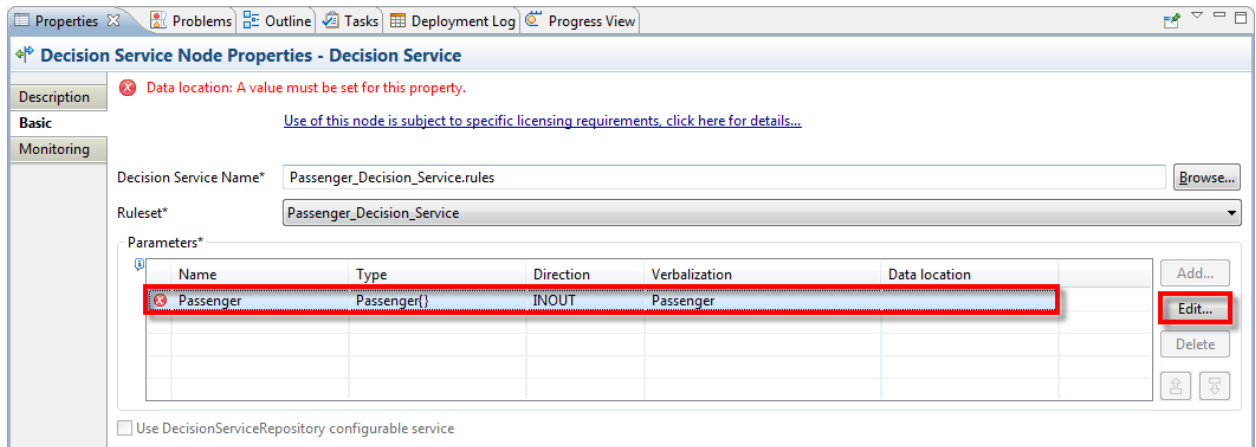
- __6. Now drag the **Passenger_Decision_Service.rules** file from the Application Development view right on to the message flow editor, and drop it after the **PASSENGER.IN** node.



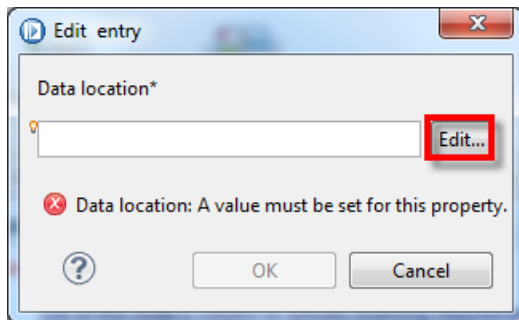
- __7. This adds a Decision Service node to the message flow, and sets the properties to use the rules file you dragged and dropped.



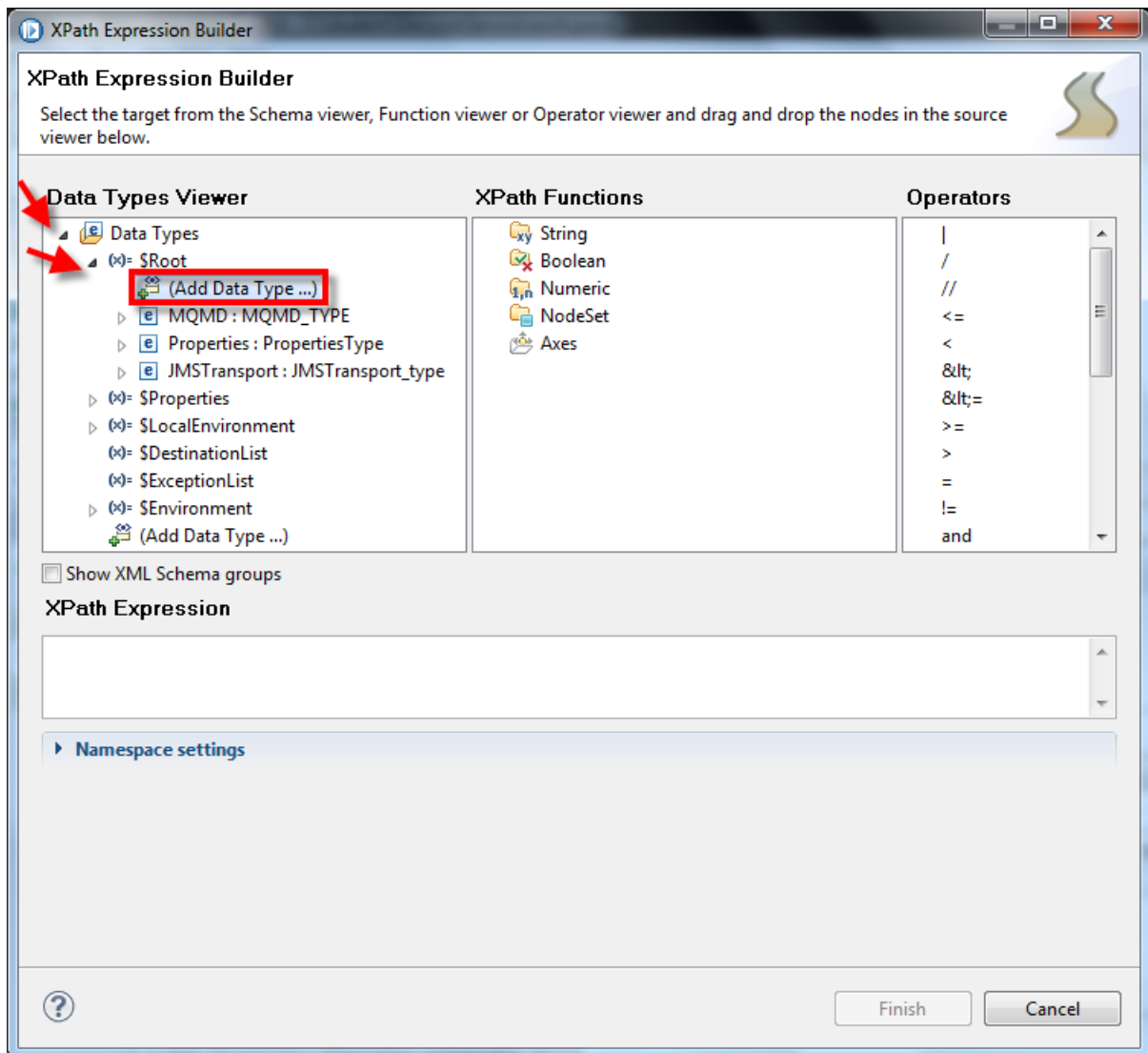
- __8. There is one property left to set. On the **Basic** tab of the Decision Service node Properties, click on the line within the Parameters box to highlight it. Then click the **Edit...** button.



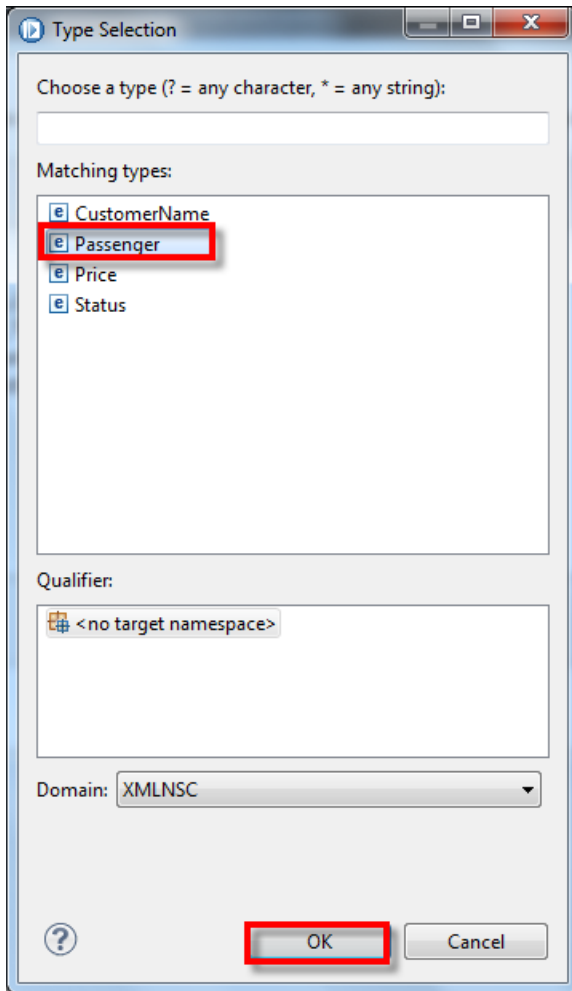
- __9. A pop-up Edit entry window appears. Click the **Edit...** button.



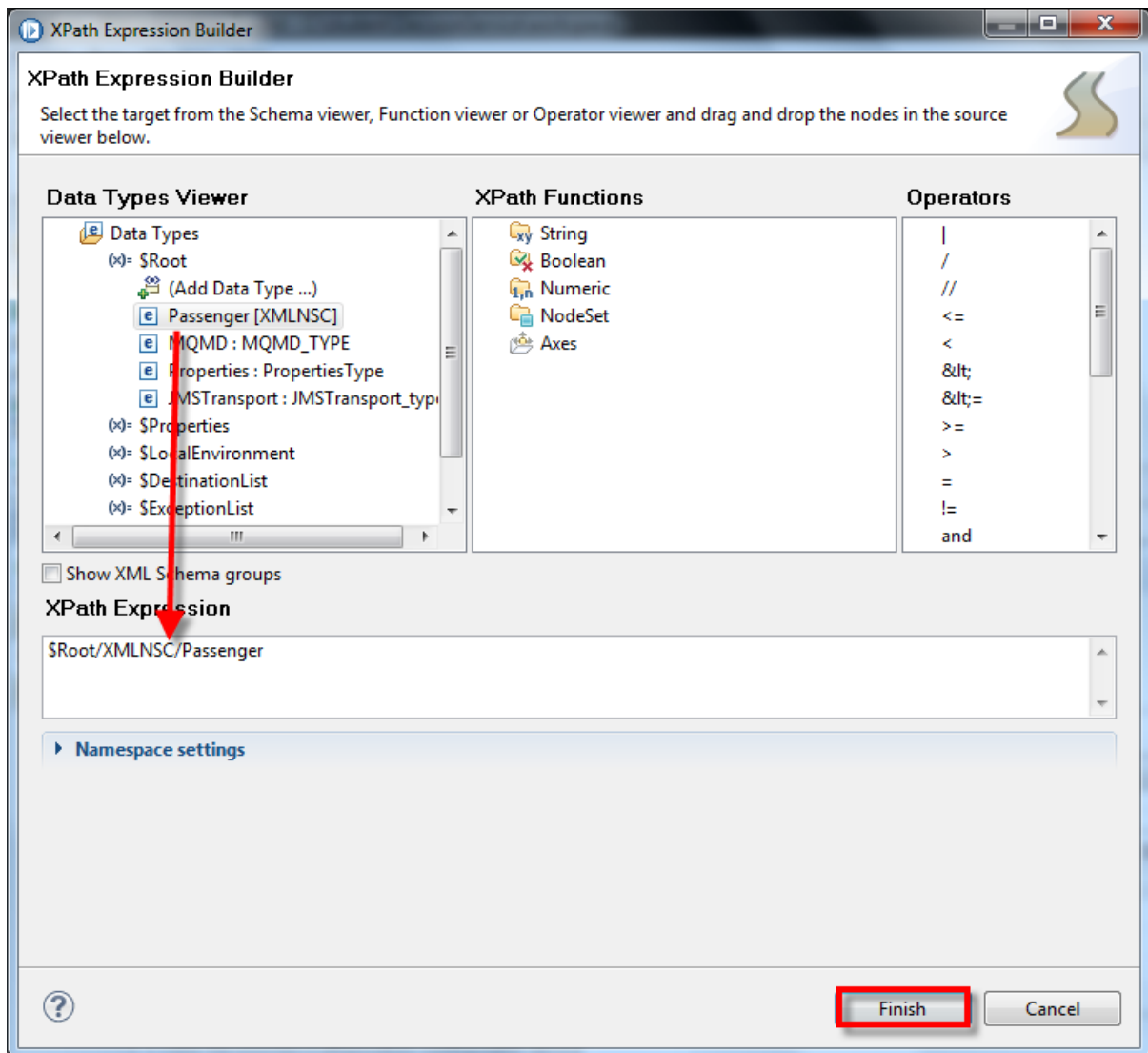
- ___10. The XPath Express Builder window opens. Click the arrow twistie next to **Data Types** and then again on **(x)= \$Root**. Click **(Add Data Type...)**.



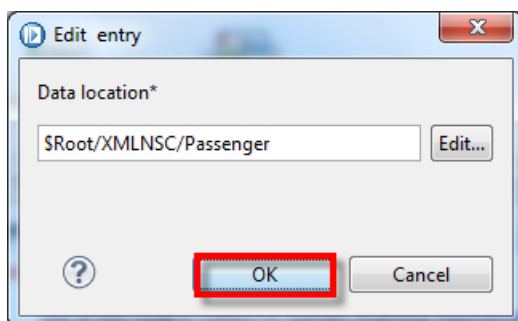
__11. On the Type Selection window that opens, select **Passenger** and click **OK**.



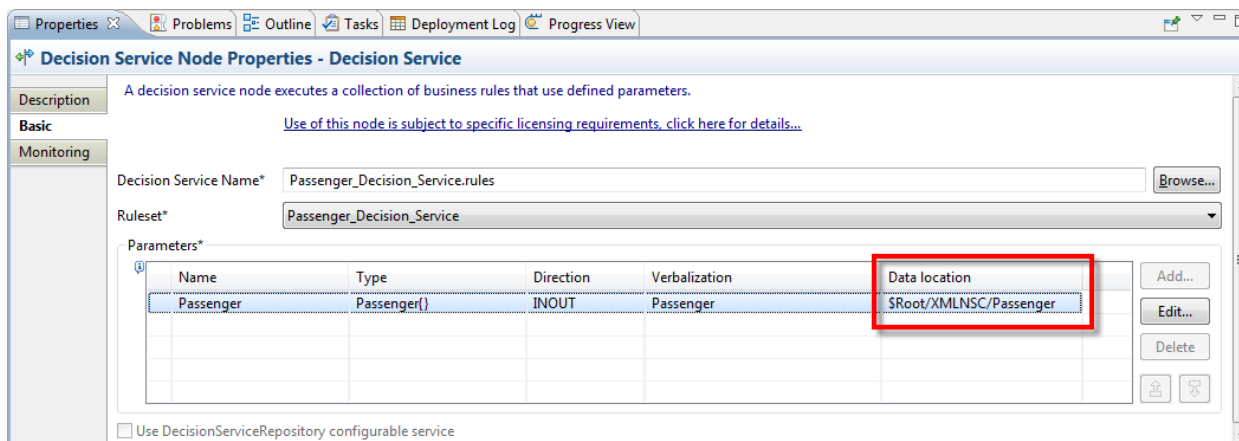
- ___12. Back on the XPath Expression Builder window, drag **Passenger** from the Data Types Viewer box down to the **XPath Expression** box. Click **Finish**.



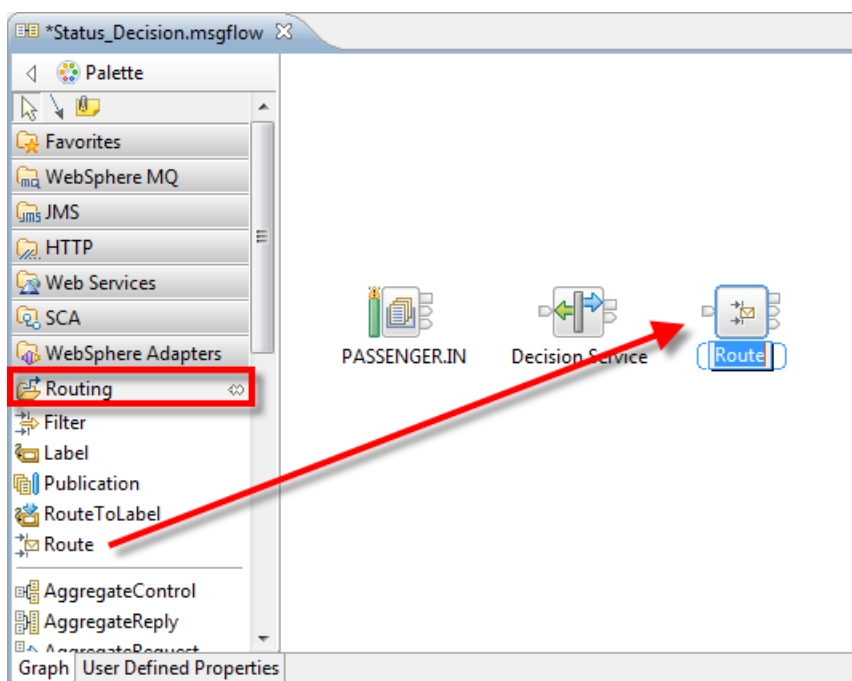
- ___13. The Data location in the Edit entry window should now be filled in with **\$Root/XMLNSC/Passenger**. Click **OK**.



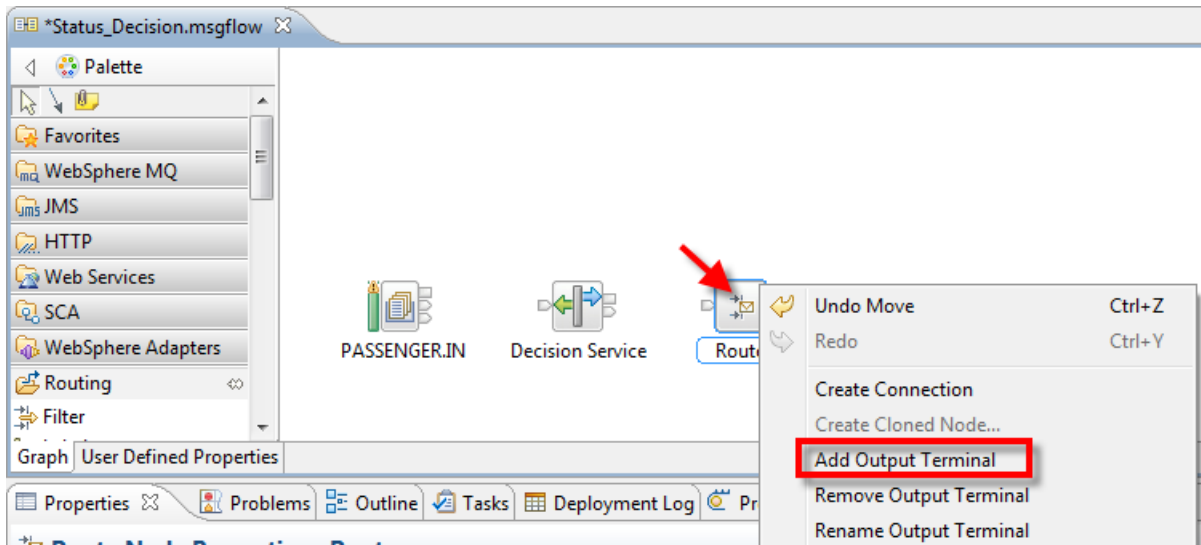
- ___14. The Data location on the Decision Service node Properties should now be filled in with **\$Root/XMLNSC/Passenger**.



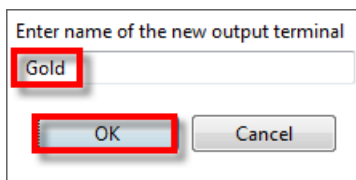
- ___15. Open the **Routing** folder in the Palette, and drag a **Route** node to the right of the Decision Service node. Press **Enter** to accept the default name.



- __16. Right-click on the **Route** node and select **Add Output Terminal**.

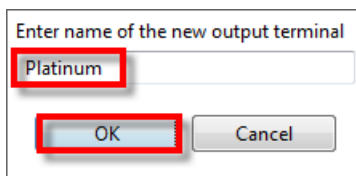


- __17. Enter **Gold** as the terminal name and click **OK**.

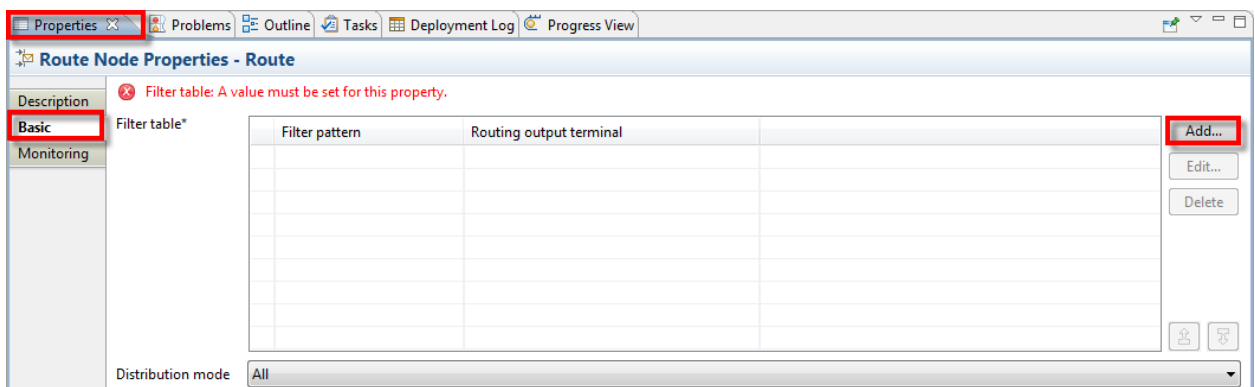


- __18. Again right-click on the **Route** node and select **Add Output Terminal**.

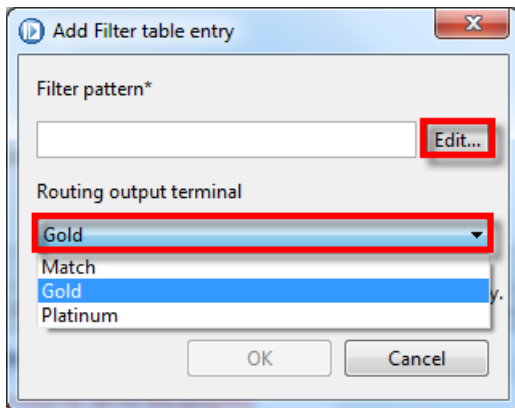
- __19. This time, enter **Platinum** as the terminal name and click **OK**.



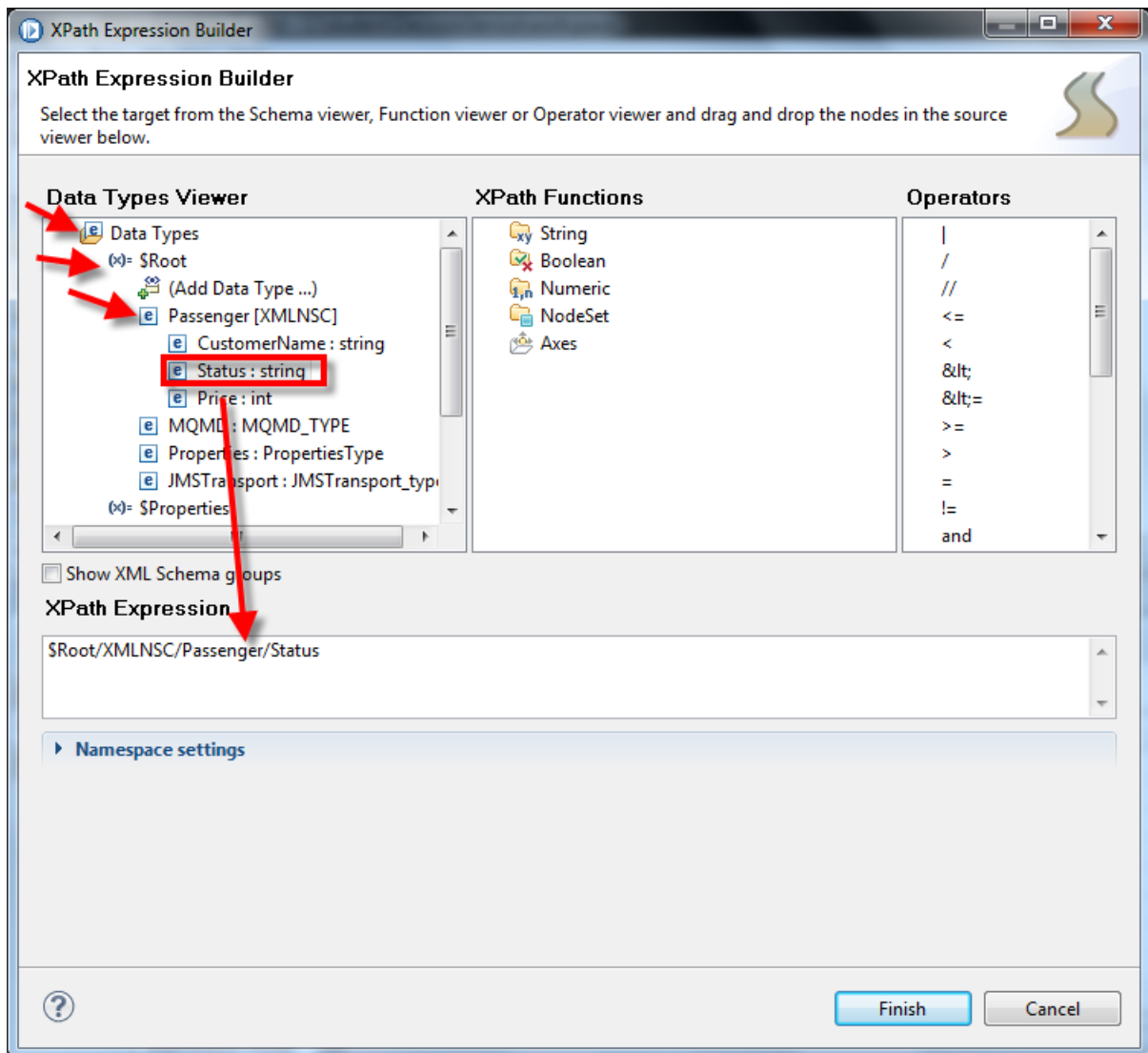
- __20. On the **Route** node Properties, on the **Basic** tab, click **Add....**



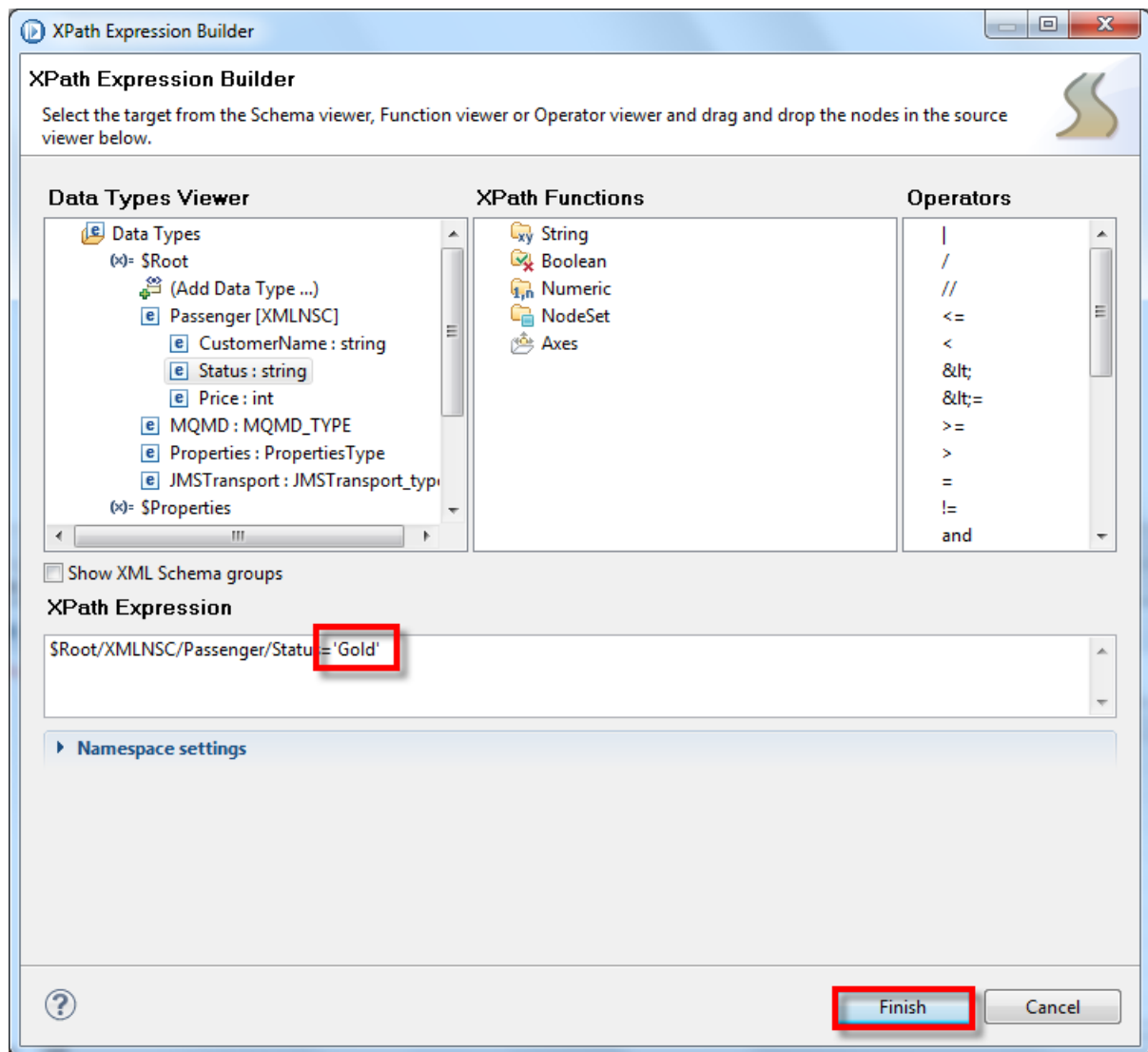
- ___21. On the Add Filter table entry window, use the drop down box for **Routing output terminal** and select **Gold**. Then click **Edit...**



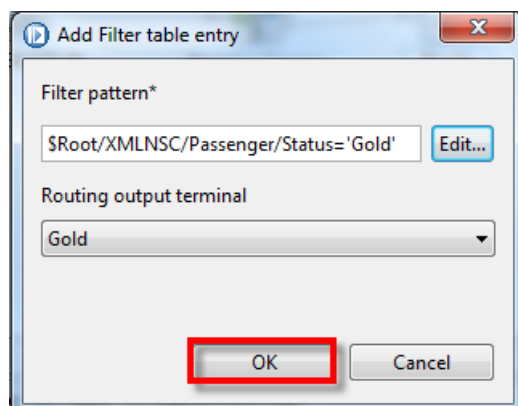
- __22. The XPath Express Builder window opens. Click the arrow twisties for **Data Types**, **(x)= \$Root**, and **Passenger**. Drag **Status** from the **Data Types Viewer** to the **XPath Expression**.



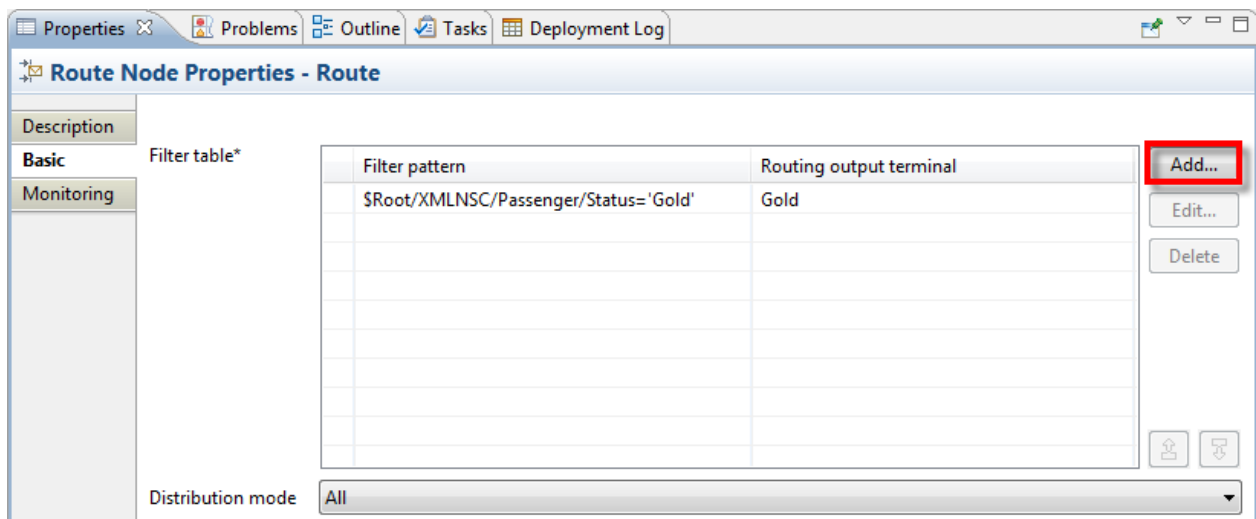
- __23. Type **'Gold'** to complete the expression in the **XPath Expression**. Click **Finish**.



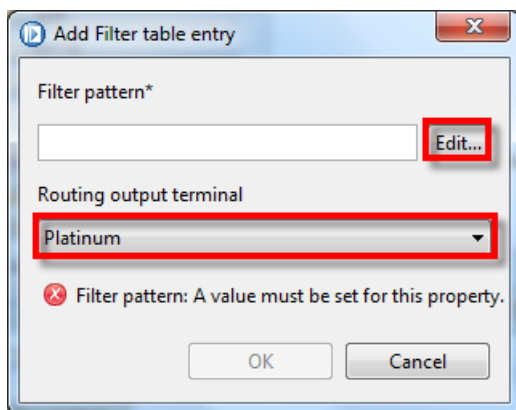
- __24. Ensure the Filter pattern reads **\$Root/XMLNSC/Passenger/Status='Gold'** and the terminal is set to **Gold**, then click **OK**.



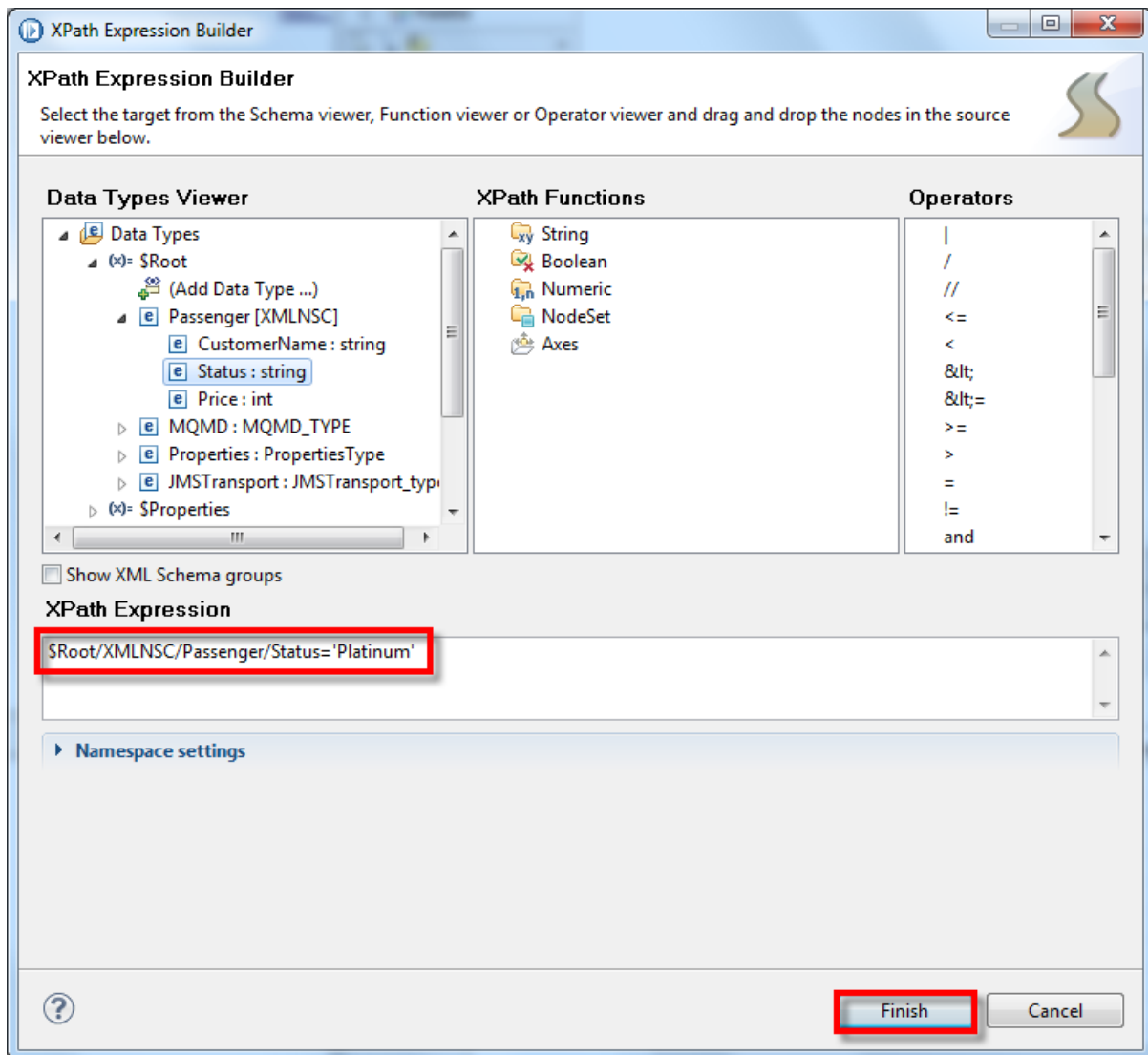
__25. The routing filter is added to the Filter table in the Properties. Click **Add...** again.



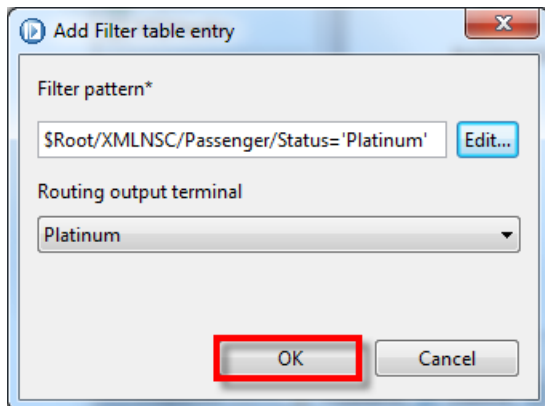
__26. This time, select **Platinum** from the drop down list of terminals, and click **Edit...**



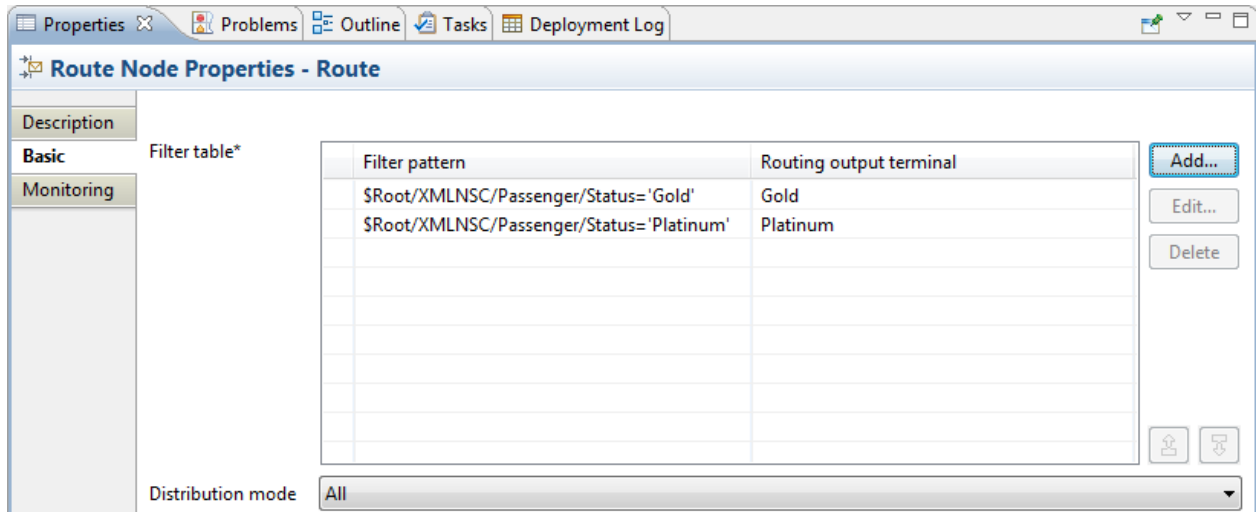
- ___27. Repeat step 22 and complete the expression in the **XPath Expression** to read **\$Root/XMLNSC/Passenger/Status='Platinum'**. Click **Finish**.



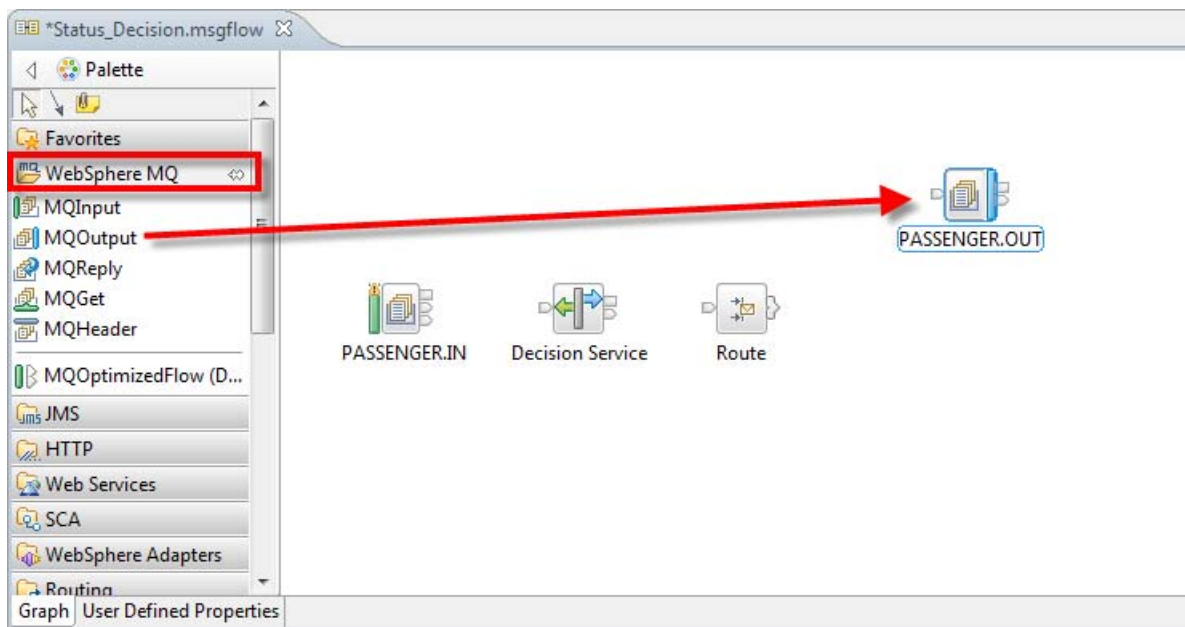
- ___28. Ensure the Filter pattern is **\$Root/XMLNSC/Passenger/Status='Platinum'** and the terminal is set to **Platinum**. Click **OK**.



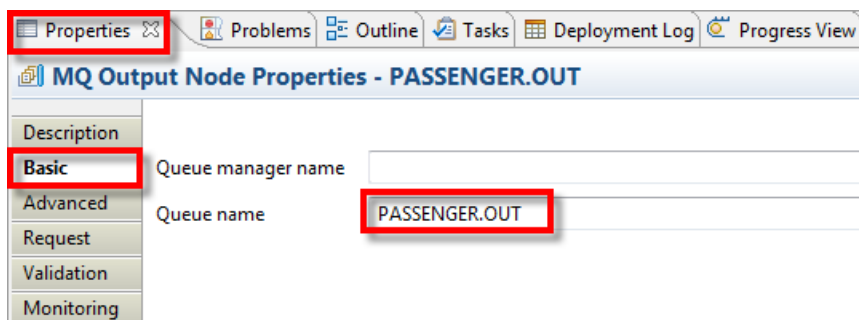
- ___29. The Route node Filter table entries are complete.



- ___30. Open the **WebSphere MQ** drawer in the Palette. Drag a **MQOutput** node to the message flow above and to the right of the Route node. Name the node **PASSENGER.OUT**.

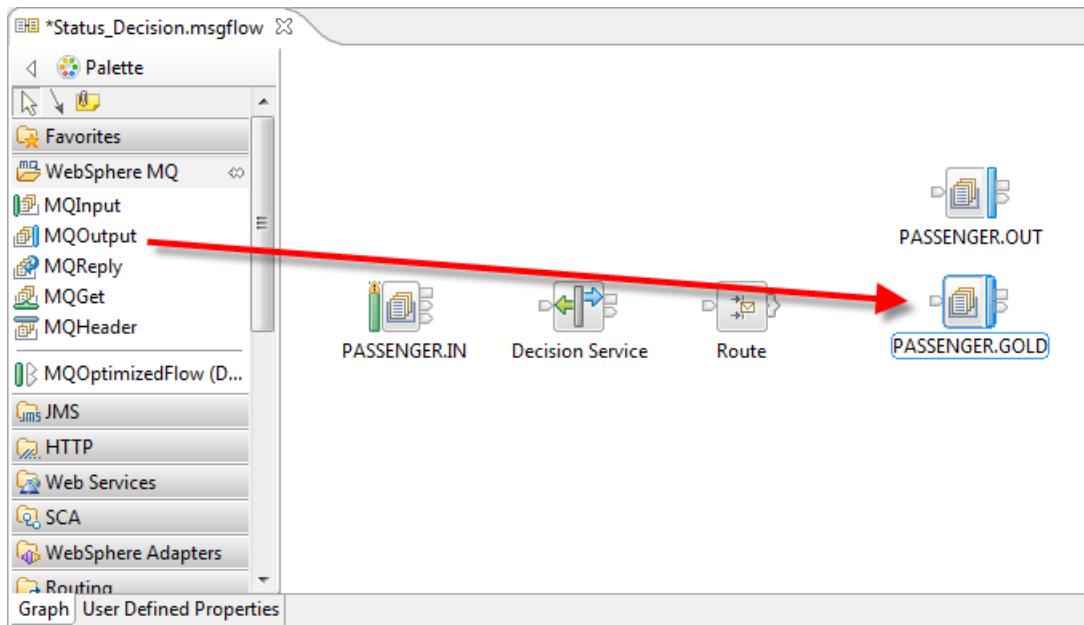


- ___31. On the MQOutput node Properties, on the **Basic** tab, set **Queue name** to **PASSENGER.OUT** (be sure you set Queue name and NOT Queue manager name).

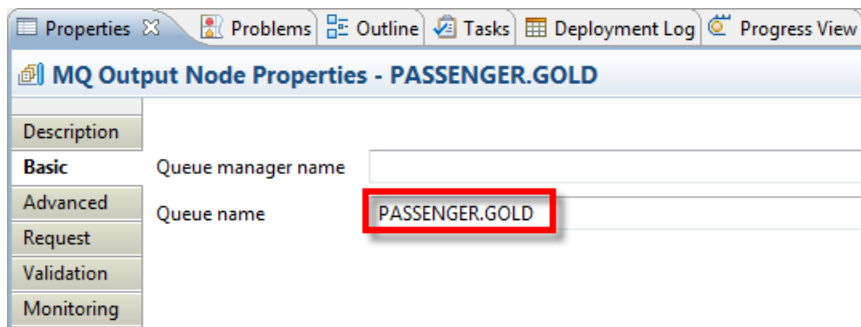


NOTE: Queue names are case sensitive.

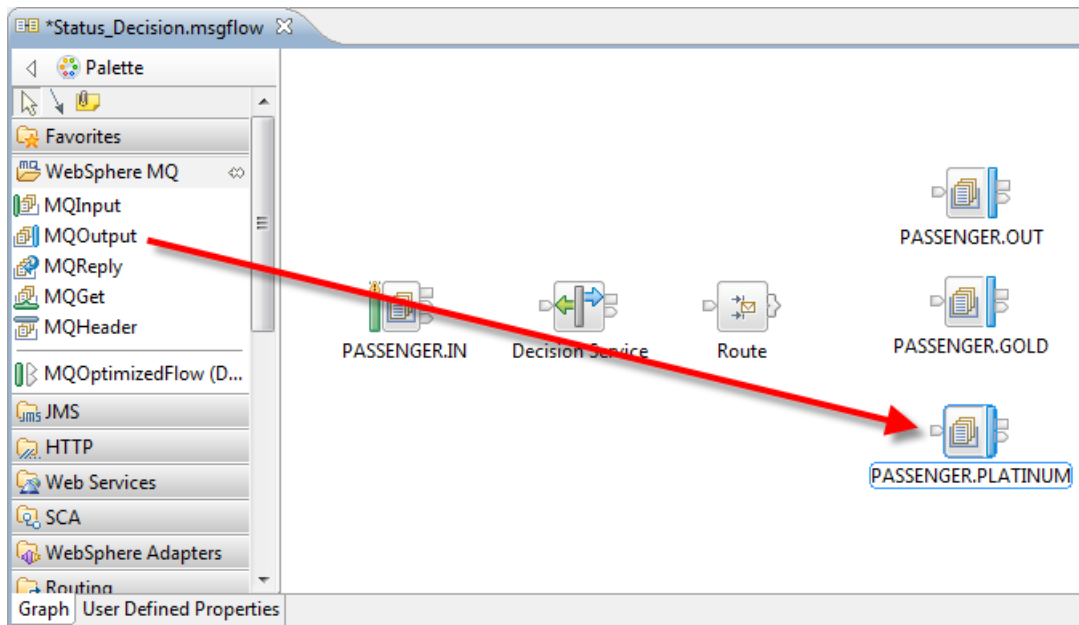
- __32. Drag another **MQOutput** node to the message flow below PASSENGER.OUT, and name the node **PASSENGER.GOLD**.



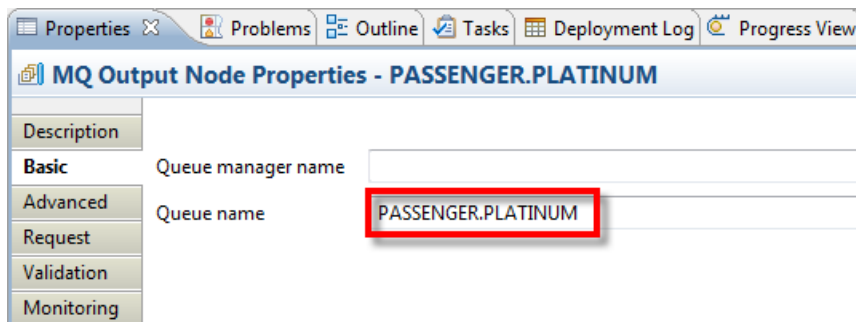
- __33. In Properties, set the Queue name to **PASSENGER.GOLD**



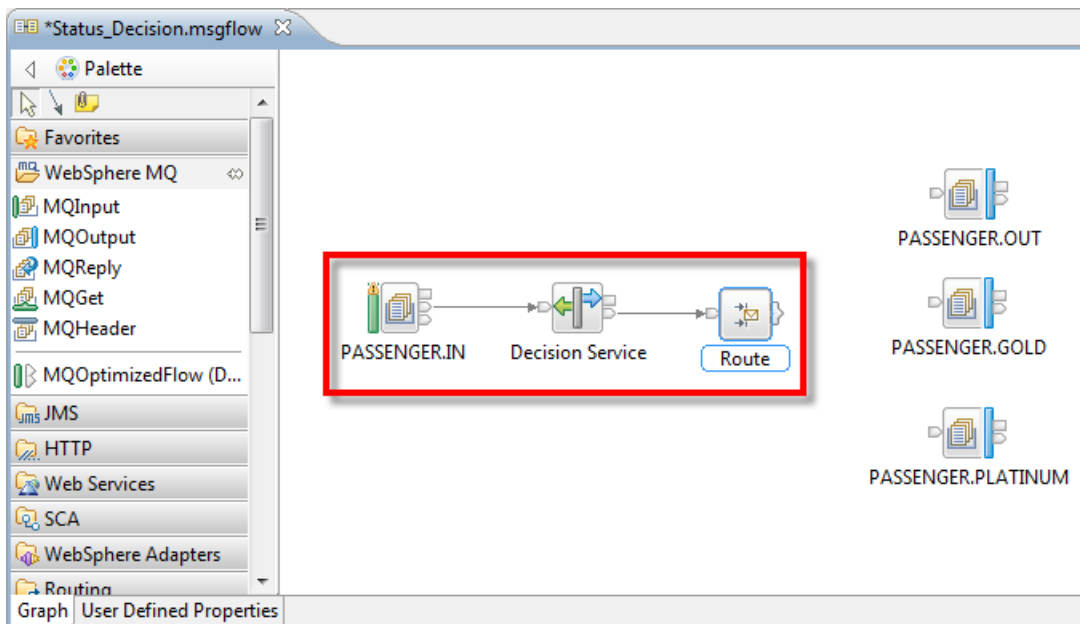
- __34. Drag a third **MQOutput** node to the message flow under the other two MQOutput nodes. Name the node **PASSENGER.PLATINUM**.



- __35. In Properties, set the Queue name to **PASSENGER.PLATINUM**

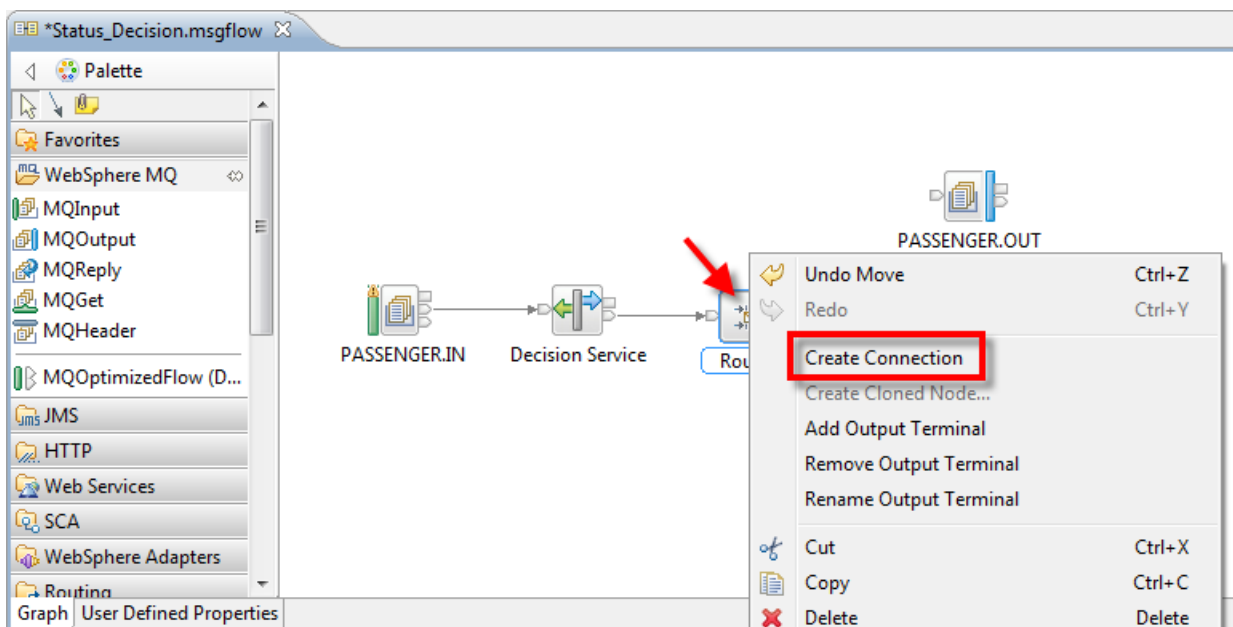


- ___36. Wire a connector from the **Out** terminal of the **PASSENGER.IN** node to the **In** terminal of the **Decision Service** node. Wire a connector from the **Out** terminal of the **Decision Service** node to the **In** terminal of the **Route** node.

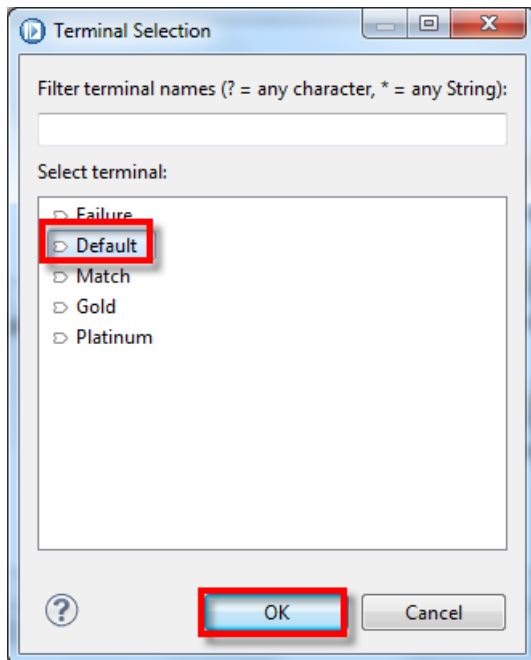


Note: To create a connector wire, you can simply click directly on the source terminal, drag the wire, and click again on the target terminal.

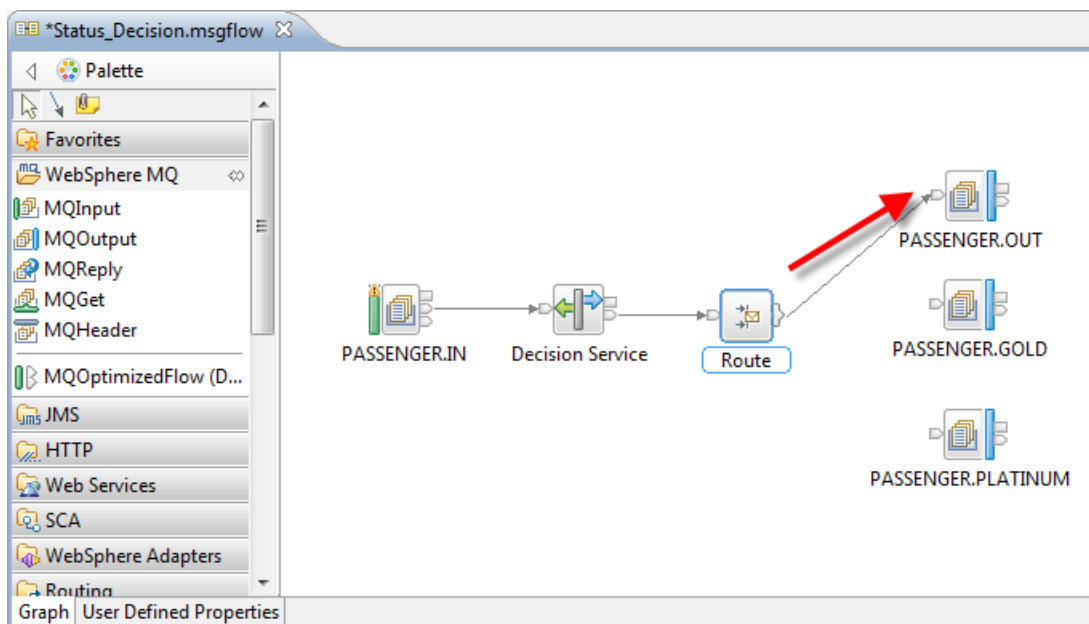
- ___37. Right-click the **Route** node and select **Create Connection** from the pop-up menu.



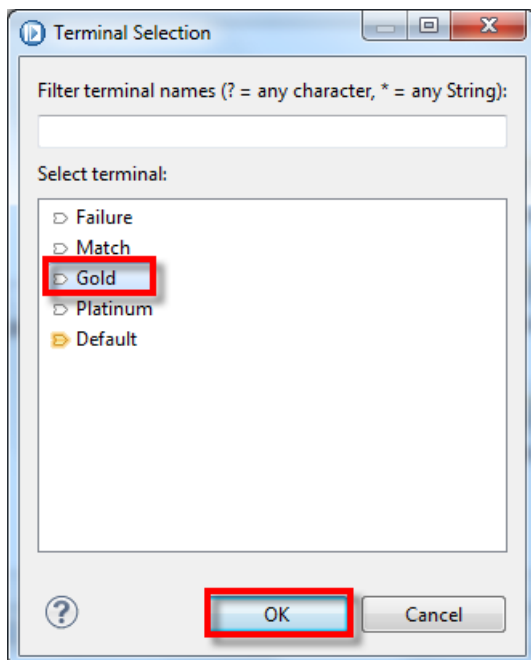
- ___38. Select the **Default** terminal from the list of terminals in the Terminal Selection window, and click **OK**.



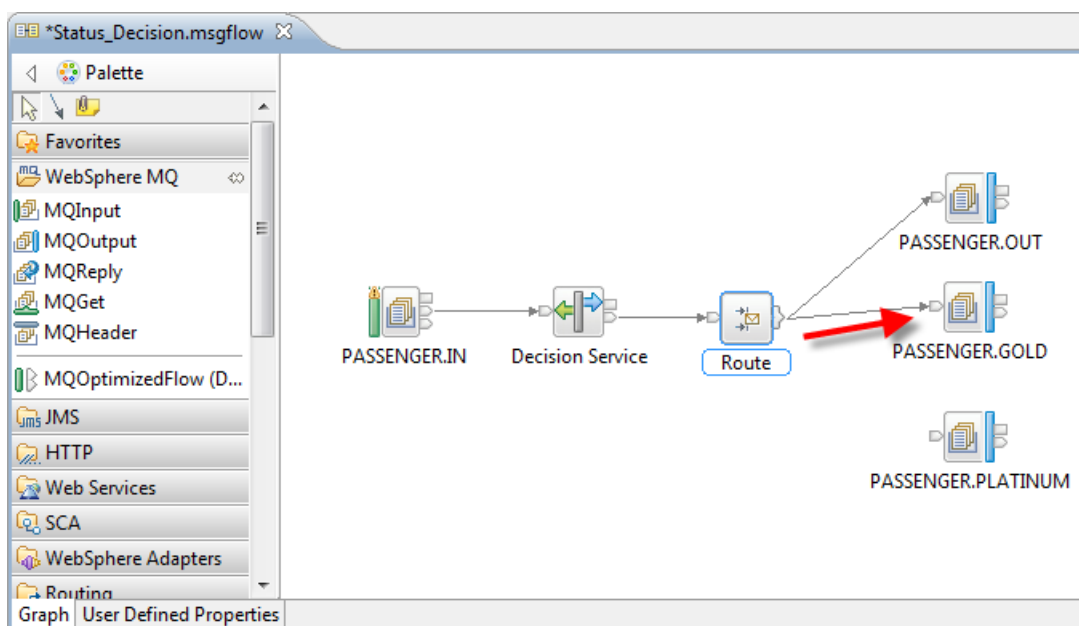
- ___39. Drag the connector to the **In** terminal of the **PASSENGER.OUT** node.



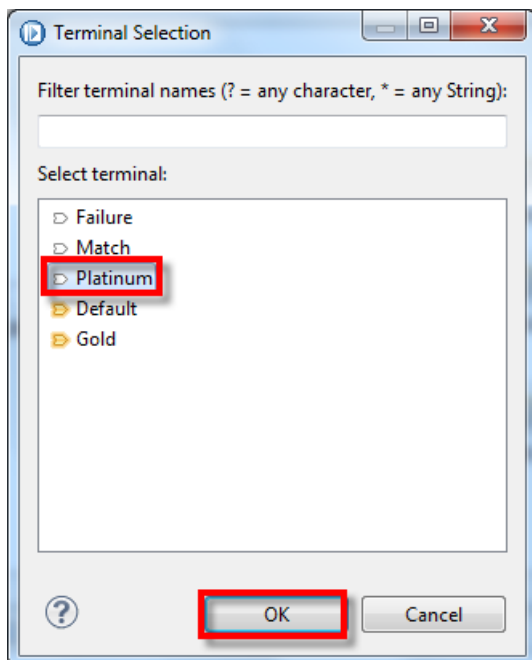
- ___40. Again right-click the **Route** node, select **Create Connection**, select the **Gold** terminal, and then click **OK**.



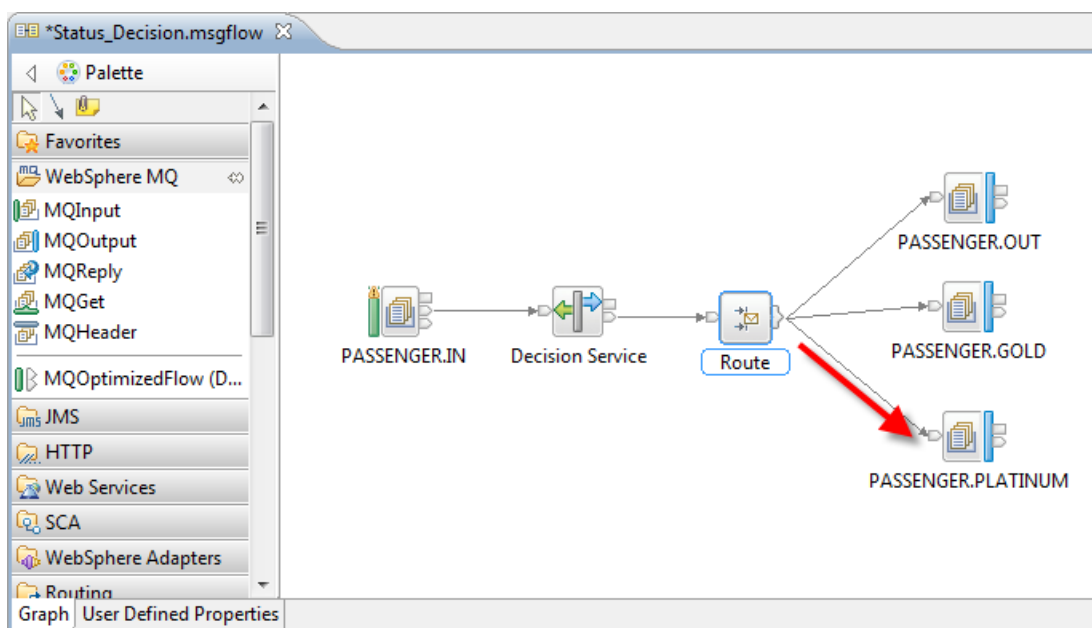
- ___41. Drag the connector to the **In** terminal of the **PASSENGER.GOLD** node.



- __42. Once more right-click the **Route** node, select **Create Connection**, select the **Platinum** terminal, and then click **OK**.



- __43. Drag the connector to the **In** terminal of the **PASSENGER.PLATINUM** node.

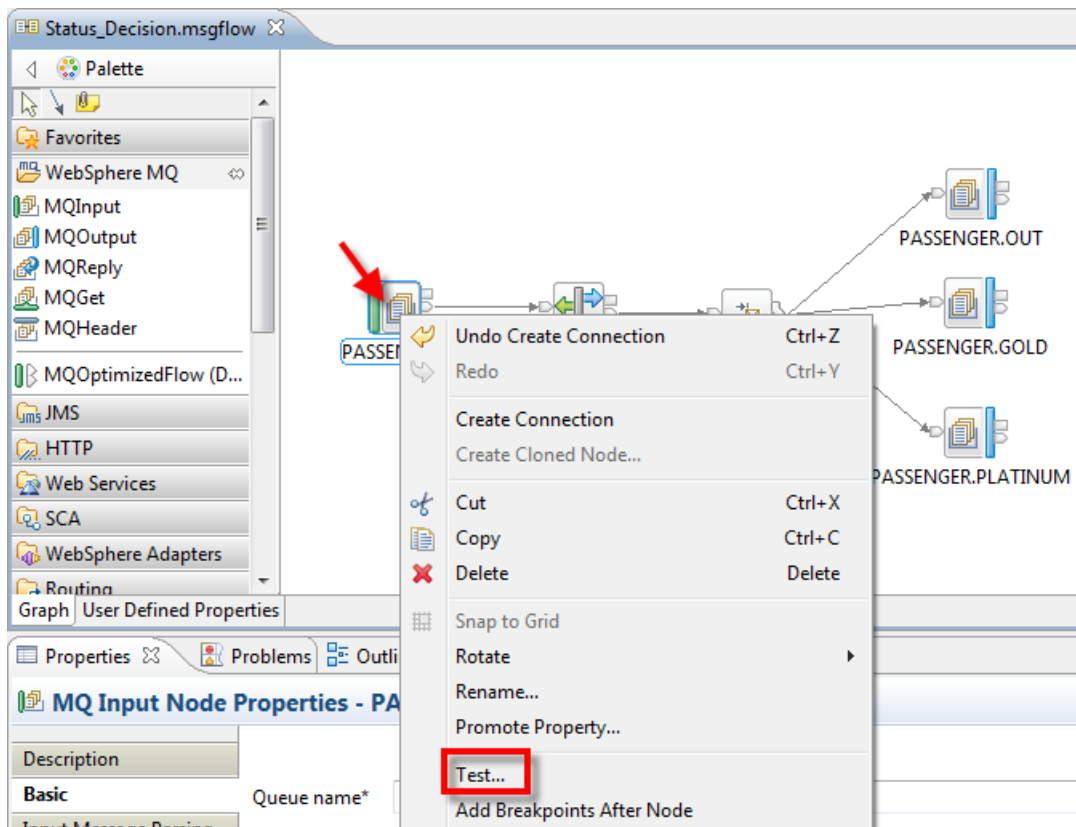


- __44. The message flow is now complete. Save (**CTRL-S**) the message flow.

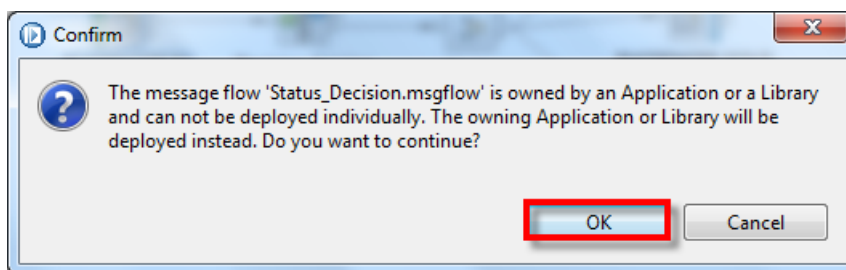
4.5 Test the Decision Service message flow

You will now test the paths of the message flow, and see that the Decision Service node executes the business rules to evaluate the inputs and set the value of the **status** element if appropriate. The **status** element is then used to make the routing decision in the Route node.

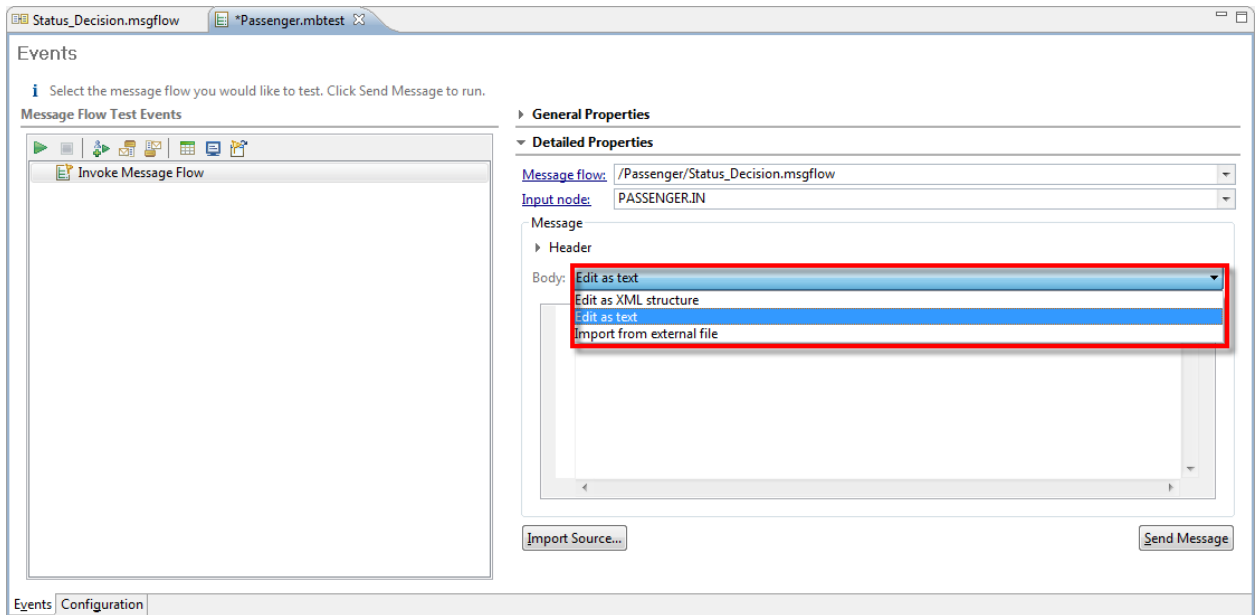
- ___1. Right-click the **PASSENGER.IN** MQInput node and select **Test...** from the popup menu.



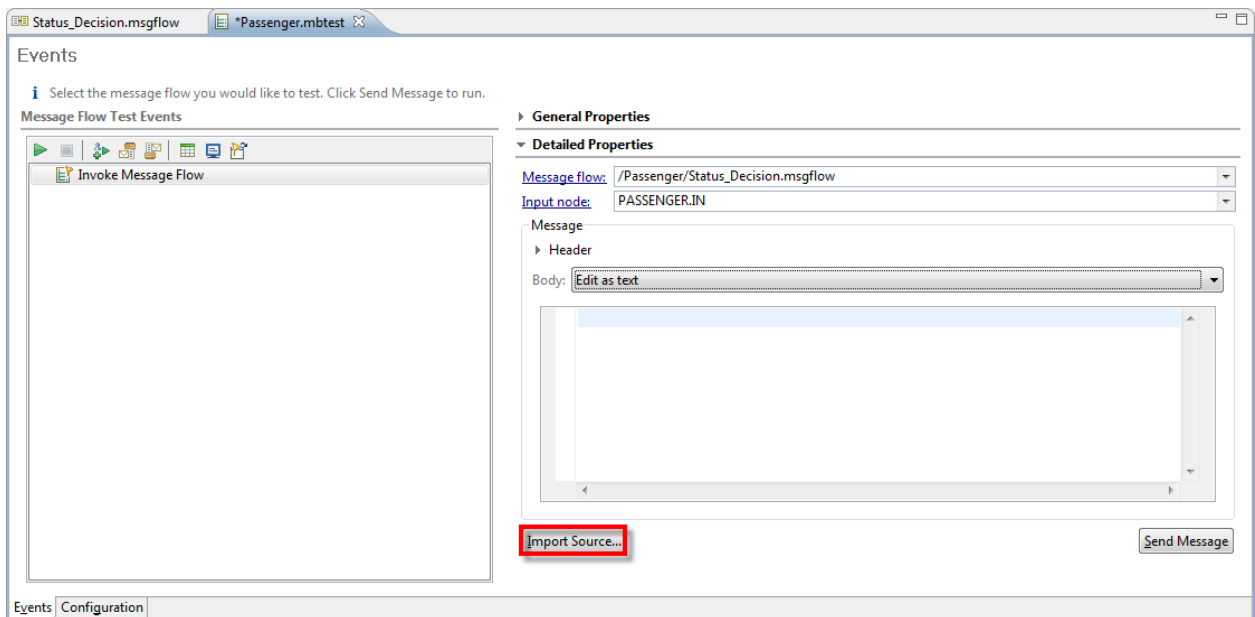
- ___2. Click **OK** on the Confirm box.



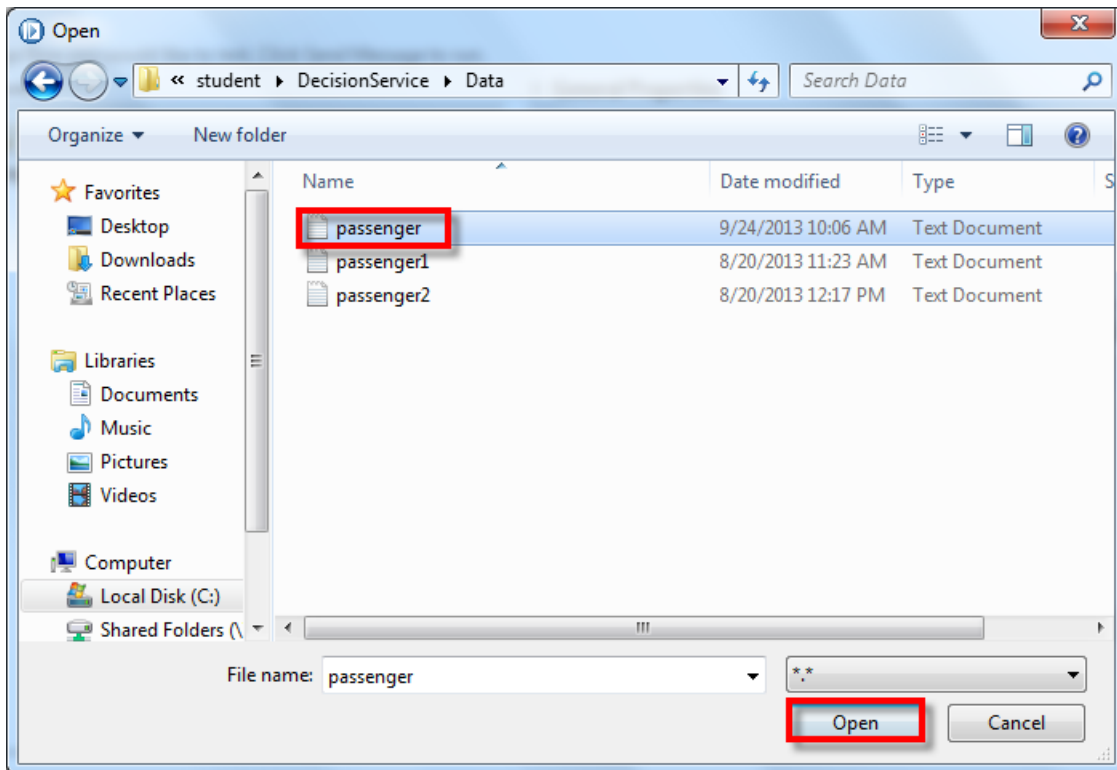
- __3. When the Integrated Test Client opens, use the drop-down for **Body** and select **Edit as text**.



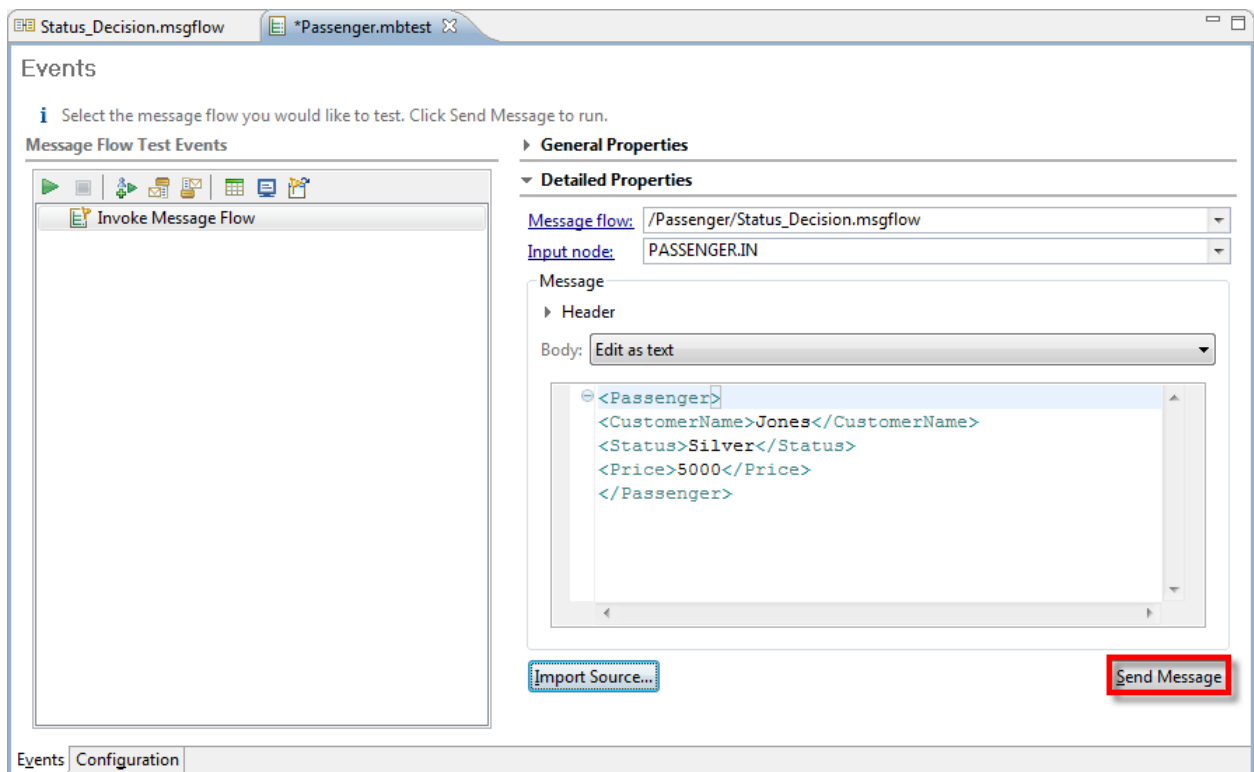
- __4. Click **Import Source...**



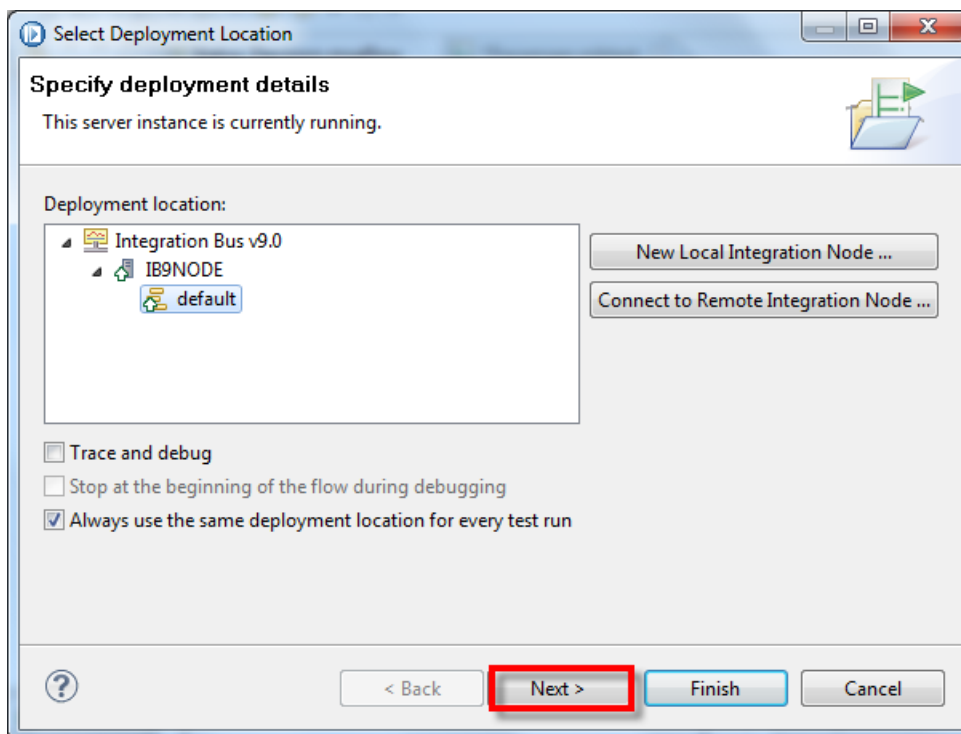
- ___5. Change to the **C:\student\DecisionService\Data** directory. Select the **passenger.txt** file and click **Open**.



__6. The test file is loaded and displayed. Click **Send Message**.



__7. Click **Next** on the Select Deployment Location window.



- __8. Set the **Seconds to wait for deployment completion** to **90** and click **Finish**.

The screenshot shows a Windows-style dialog box titled "Select Deployment Location". Inside, there is a section titled "Specify Test Settings" with the instruction "Determine the settings of the test environment." Below this, there are several settings:

- "Seconds to wait for deployment completion" is set to 90 (highlighted with a red box).
- "Seconds to wait after Test Client complete the deployment" is set to 0.
- "Seconds to wait on launching the debugger for tracing purpose" is set to 20.
- A checkbox "Show information dialog before disconnecting debugger." is unchecked.
- "Seconds to wait for test client to stop" is set to 120.
- A checkbox "Create queues of input and output nodes of message flows when host name is localhost" is unchecked.
- A checkbox "Add or modify (but not clear) what has already been deployed on the execution group" is unchecked.

At the bottom of the dialog, there are four buttons: "< Back", "Next >", "Finish" (highlighted with a red box), and "Cancel". A help icon (?) is also present in the bottom left corner.

9. The Passenger Application is packaged into a BAR file (created and stored in Independent Resources→TestClientBarFiles) and then deployed to the default Integration Server. The test message is then sent to the PASSENGER.IN MQInput node, which is then processed by the Status_Decision message flow. You see the message sent to PASSENGER.IN in the test Events, and then you see the Event when the message is sent to one of the three MQOutput nodes. Which MQOutput node the message is sent to depends on the decision made and message Status element being updated.

For this test, the message is routed to the PASSENGER.OUT MQOutput node; therefore, the business rules that were executed did not alter the Status, as no rules matched, and the Route node sent the message to the default terminal, which is wired to PASSENGER.OUT.

The screenshot displays the IBM Integration Bus test environment. The top window shows two tabs: 'Status_Decision.msgflow' and '*Passenger.mbttest'. The main area is titled 'Events' and contains a 'Message Flow Test Events' pane on the left and a 'General Properties' pane on the right.

The 'Message Flow Test Events' pane shows a sequence of events:

- Invoke Message Flow
- Message flows deployment successfully completed
- Starting
- Sending Message to MQ Queue "PASSENGER.IN"
- MQ Queue Monitor "PASSENGER.OUT"** (highlighted with a red box)
- Stopped listening for response
- Stopped

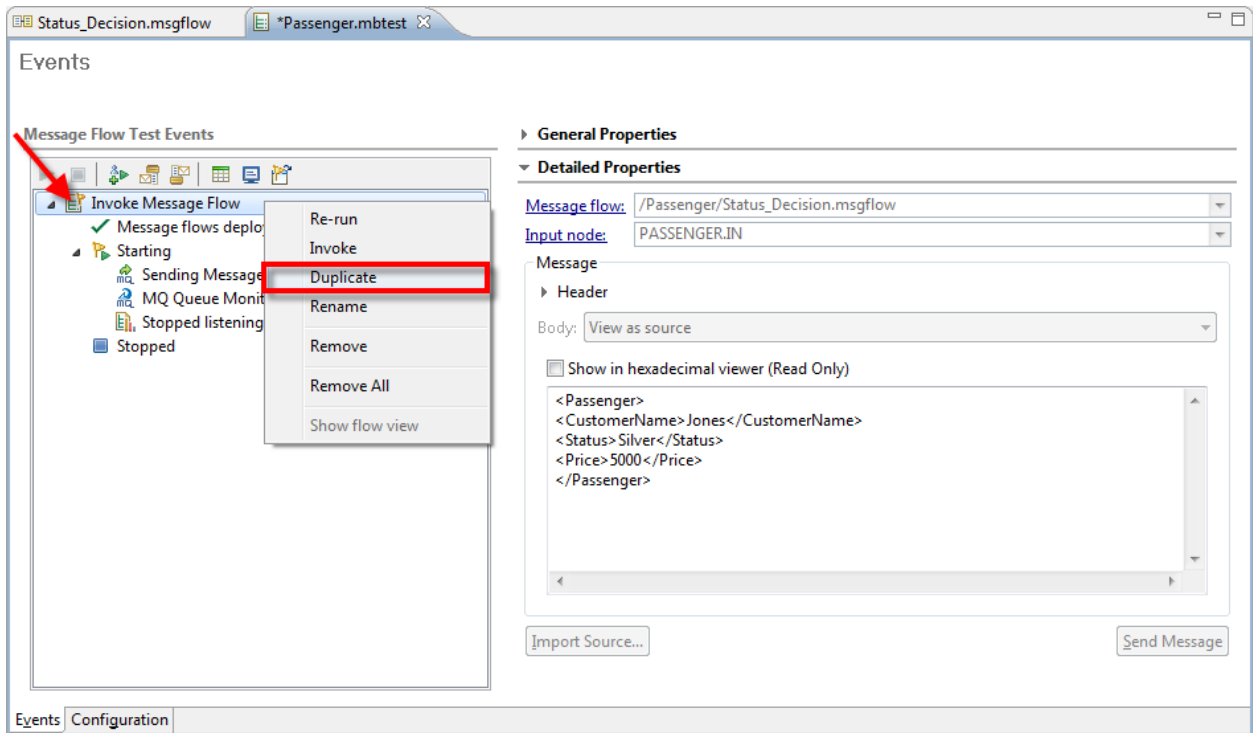
The 'General Properties' pane shows the following configuration:

- Host: localhost
- Port: 0
- Server channel: SYSTEM.BKR.CONFIG
- Queue manager: IB9QMGR
- Queue: PASSENGER.OUT

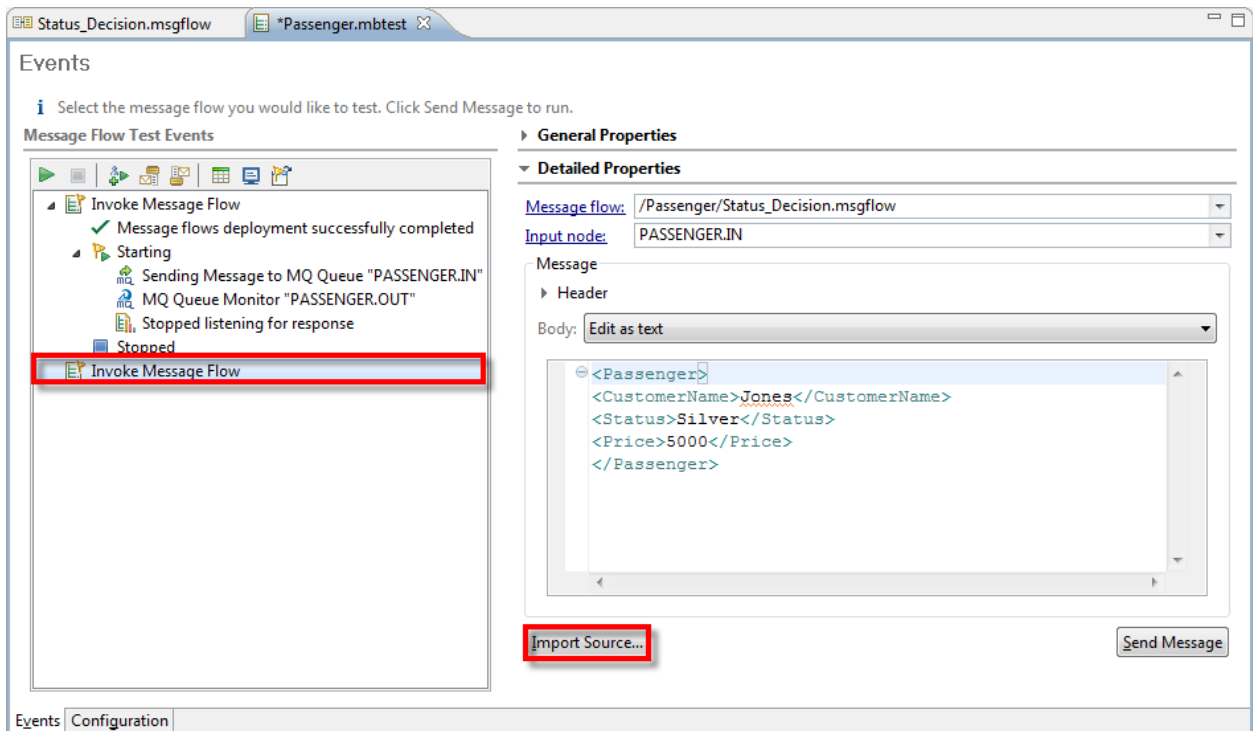
The 'Message' section shows a 'Header' and a 'Body' (View as XML structure). The body contains a table with the following data:

Name	Value
Passenger	
CustomerName	Jones
Status	Silver
Price	5000

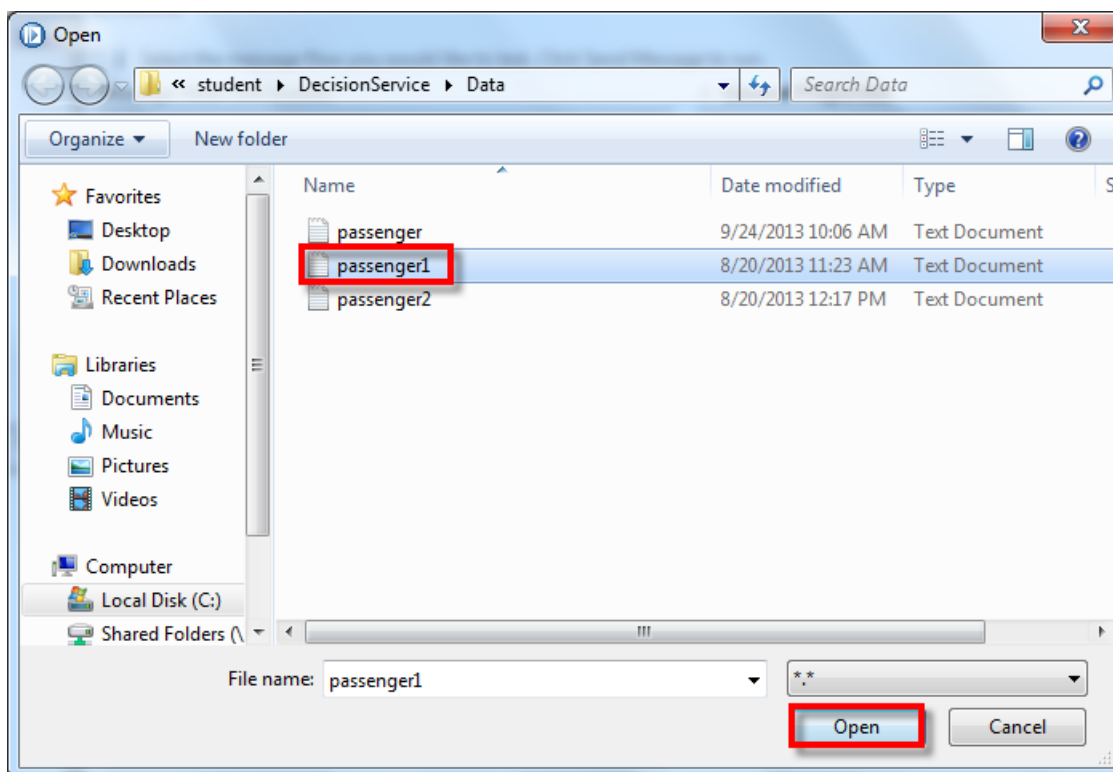
- __10. Right-click on the **Invoke Message Flow** event, and select **Duplicate** from the pop-up menu.



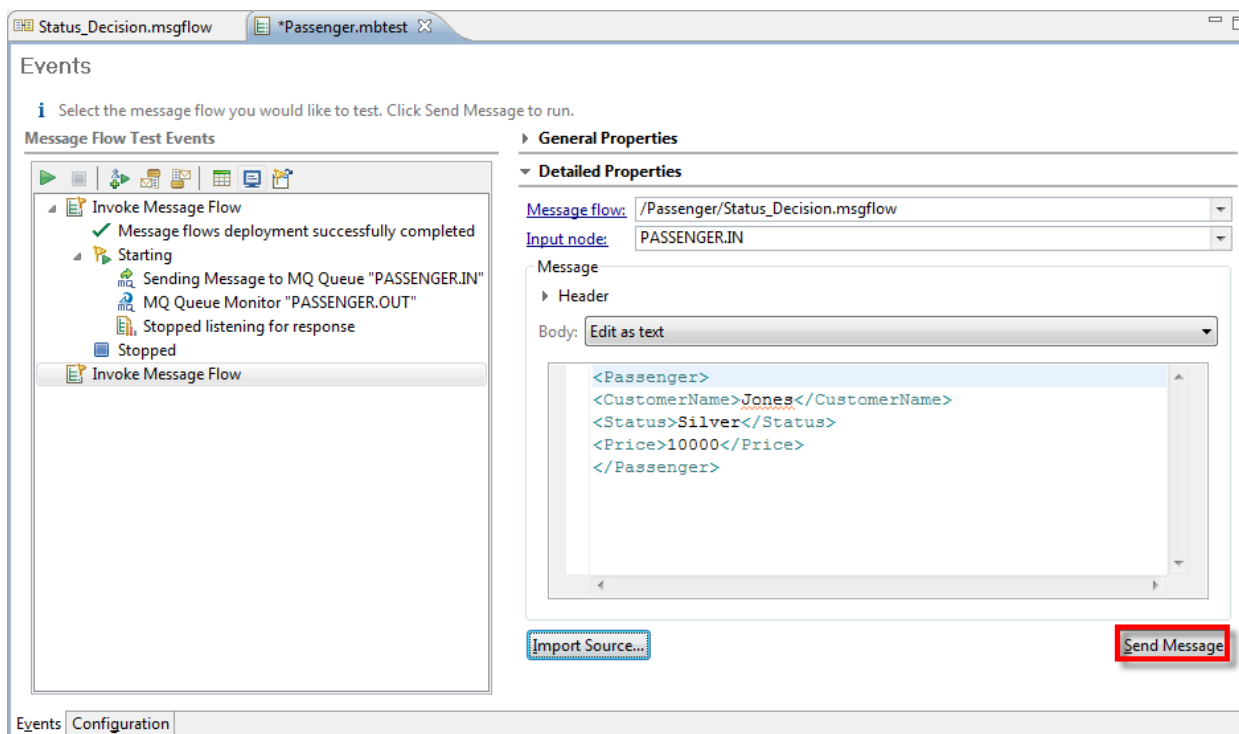
- __11. With the new (second) **Invoke Message Flow** event highlighted, click **Import Source....**



__12. Select the **passenger1.txt** file, and click **Open**.



__13. The test file is loaded. Notice the **Status** in the test message is **Silver**. Click **Send Message**.



- ___14. This time, we see the message sent to the **PASSENGER.GOLD** MQOutput node in the events, and we can see that **Status** of the message was changed to **Gold**. This was done by the successful evaluation of Rule 1 in our Decision Service node.

The screenshot shows the IBM MessageFlow Tester interface. The 'Events' tab is active, displaying a list of 'Message Flow Test Events'. The events include 'Invoke Message Flow', 'Message flows deployment successfully completed', 'Starting', 'Sending Message to MQ Queue "/>

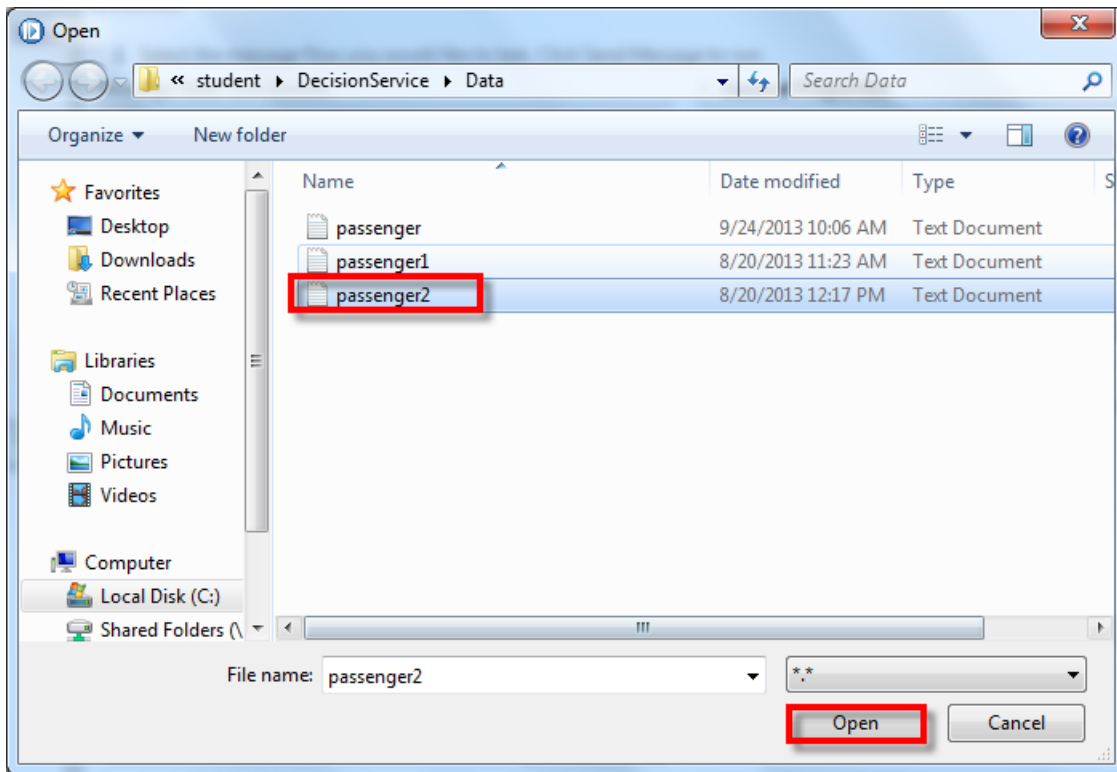
- ___15. Right-click on either **Invoke Message Flow** event, and select **Duplicate** again.

The screenshot shows the IBM MessageFlow Tester interface. The 'Events' tab is active, displaying a list of 'Message Flow Test Events'. A right-click context menu is open over the 'Invoke Message Flow' event, with the 'Duplicate' option highlighted. The 'Detailed Properties' tab on the right shows the message body in XML format.

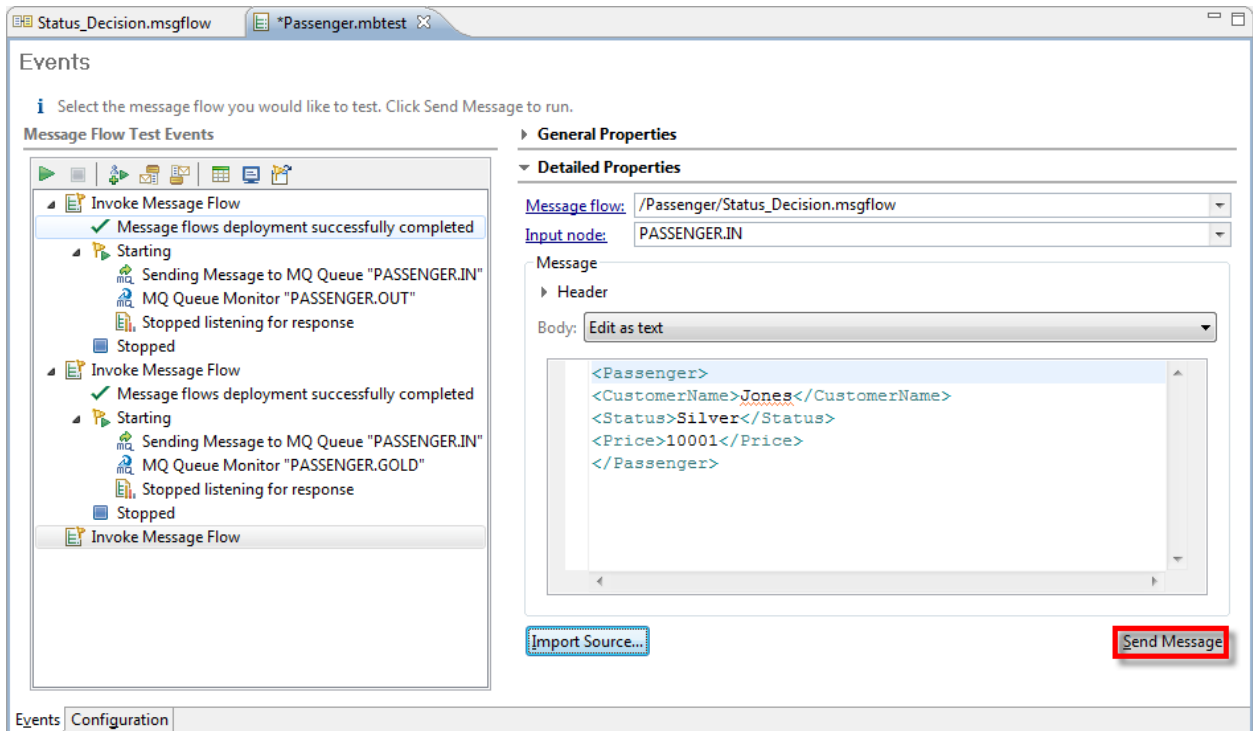
Name	Value
Passenger	
CustomerName	Jones
Status	Gold
Price	10000

```
<Passenger>
  <CustomerName>Jones</CustomerName>
  <Status>Silver</Status>
  <Price>10000</Price>
</Passenger>
```

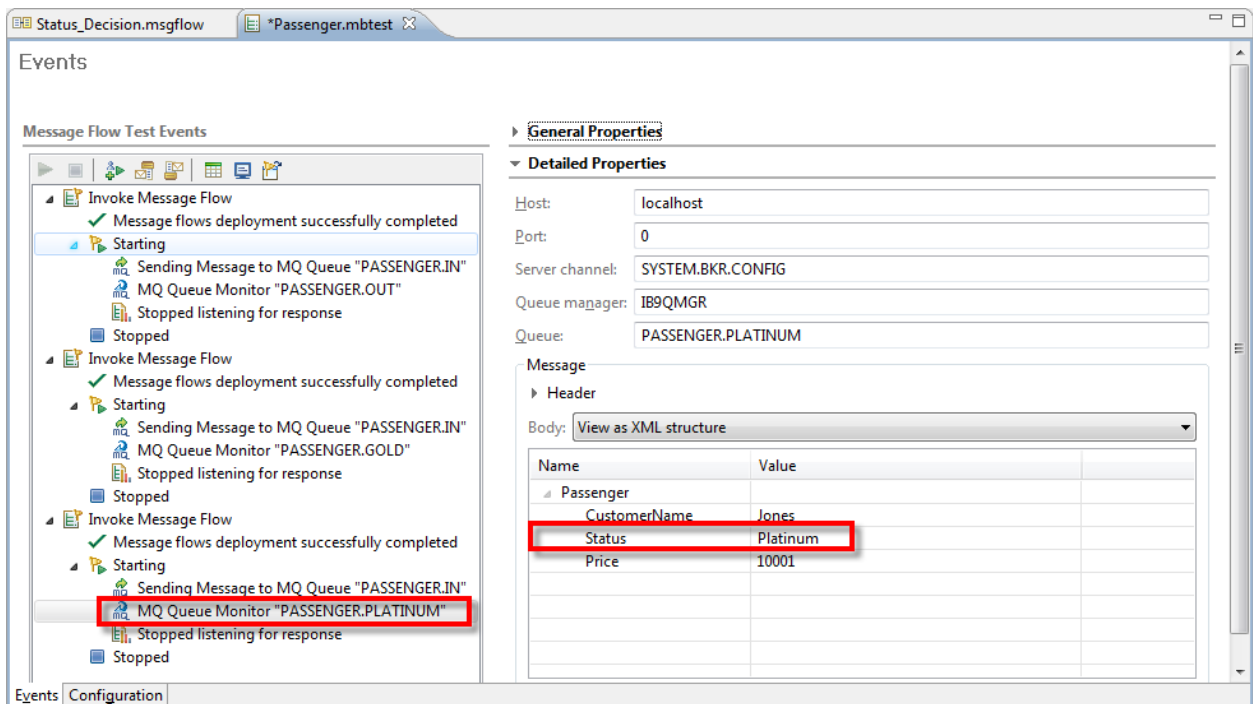
- __16. With the new (third) **Invoke Message Flow** event highlighted, click **Import Source....**
- __17. Select the **passenger2.txt** file and click **Open**.



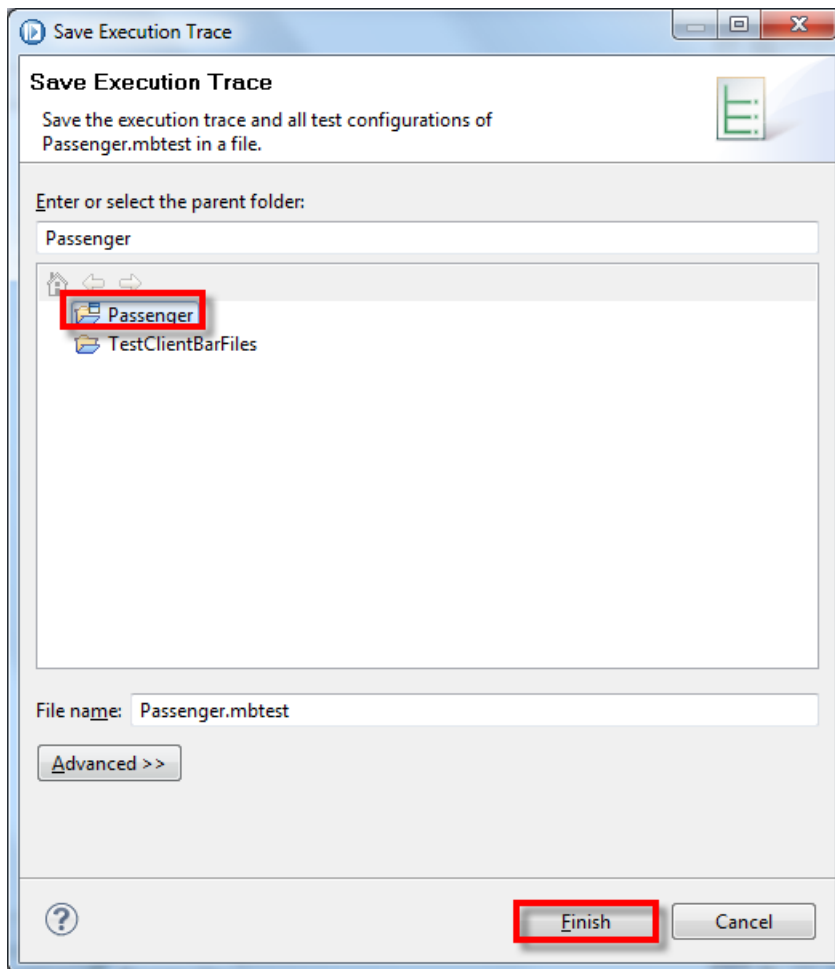
- __18. The test file is loaded. Notice the **Status** in the test message is **Silver**. Click **Send Message**.



- __19. This time, we see the message sent to the **PASSENGER.PLATINUM** MQOutput node in the events, and we can see that **Status** of the message was changed to **Platinum**. This was done by the successful evaluation of Rule 2 in our Decision Service node.



- ___20. Save the test file (**CTRL-S**), allowing all three tests to be saved for testing again. Select the **Passenger** folder, and click **Finish**.



CONGRATULATIONS!

This concludes the Decision Service node lab.

This is the end of Lab 4.

Lab 5 Service discovery for database and MQ Services

5.1 Introduction to database and MQ Services

IBM Integration Bus V9.0 provides a new service discovery feature for MQ and database resources. Discovered services are used at development time to create Integration Bus Toolkit MQ and database artifacts. These artifacts are then used to create applications. The MQ and database artifacts are then deployed to the Integration Bus runtime environment.

MQ Services can be optionally stored in an Integration Registry component which runs on an Integration Node. MQ Services stored in this way can be imported by other Integration Bus developers.

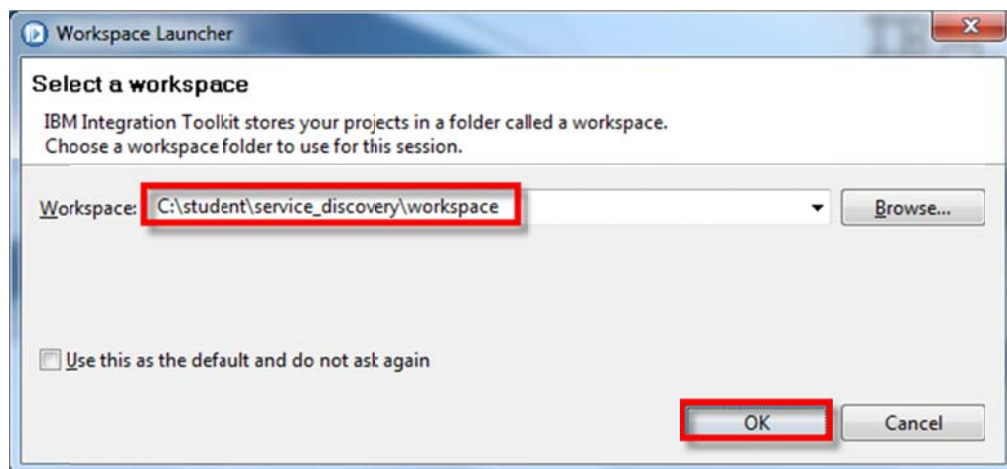
This lab guide will demonstrate:

- How to create a Request-Response MQ Service
- How to store MQ services in an Integration Registry
- How to import previously stored MQ services into a developer's workspace
- How to create a database service (*note: database services cannot currently be stored in the Integration Registry*)
- How to extend an existing database service by adding additional operations
- How to use these services at development time to create Integration Bus MQ and database artifacts
- How to use the Integration Bus artifacts to create and extend applications

5.2 Lab preparation

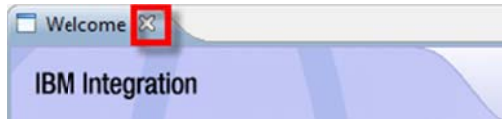
5.2.1 Priming the Integration Service Registry

- ___1. If not open, start the IBM Integration Bus Toolkit by double-clicking its desktop icon. Enter **c:\Student\service_discovery\workspace** as the workspace and select **OK**.



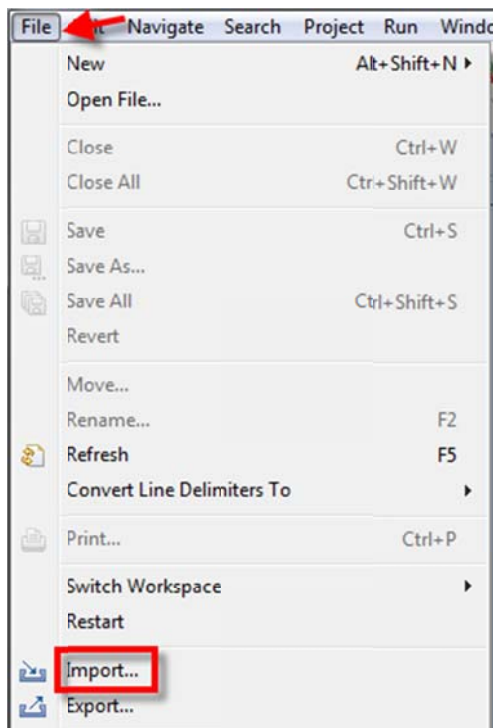
If already open, select **File→Switch Workspace→Other...** to get the workspace selection dialog, and then enter (or use Browse) **c:\Student\service_discovery\workspace** as the workspace and select **OK**.

- __2. Click the **X** to close the Welcome screen.

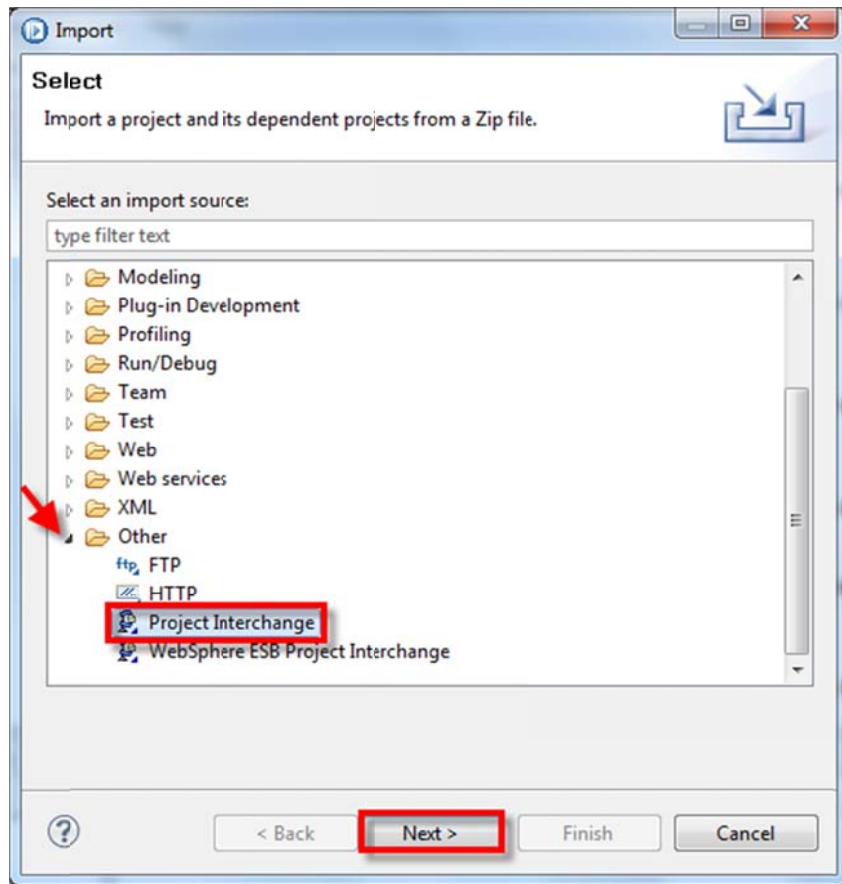


- __3. The required MQ Service is available in a Project Interchange (PI) file in **c:\student\service_discovery\install\PrimeServiceRegistry.zip**. Import this PI file into the Integration Toolkit.

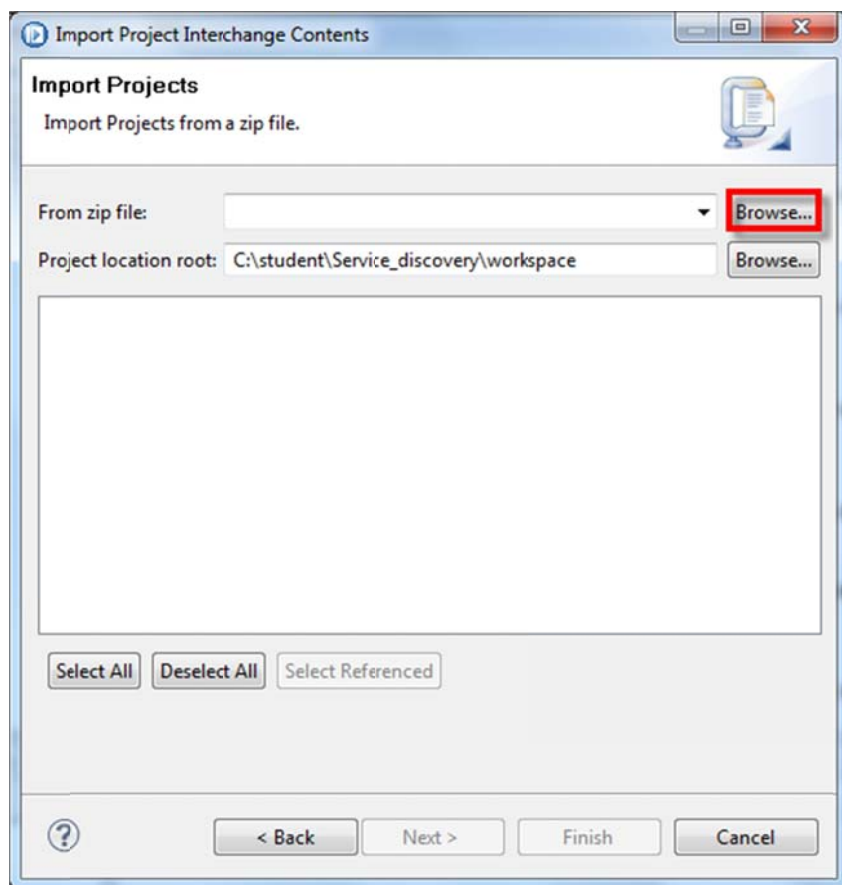
Select **File→Import...** from the menu.



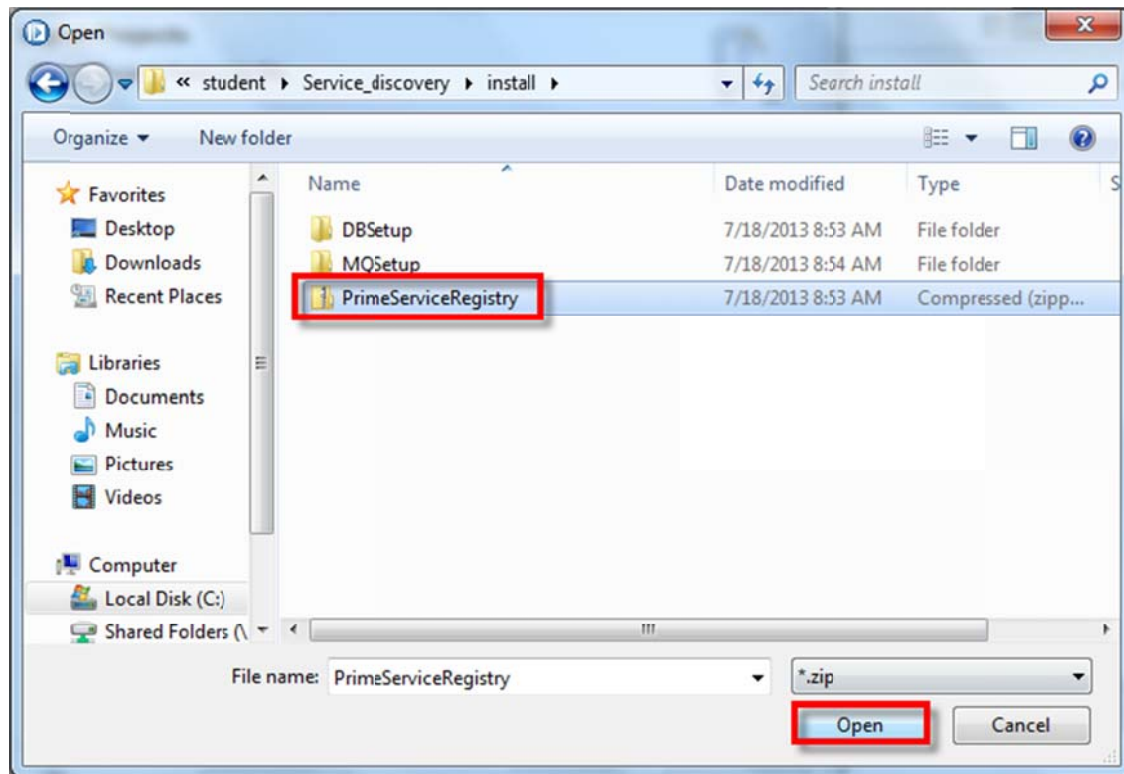
- ___4. Click the twistie arrow for the **Other** folder, select **Project Interchange**, and click **Next**.



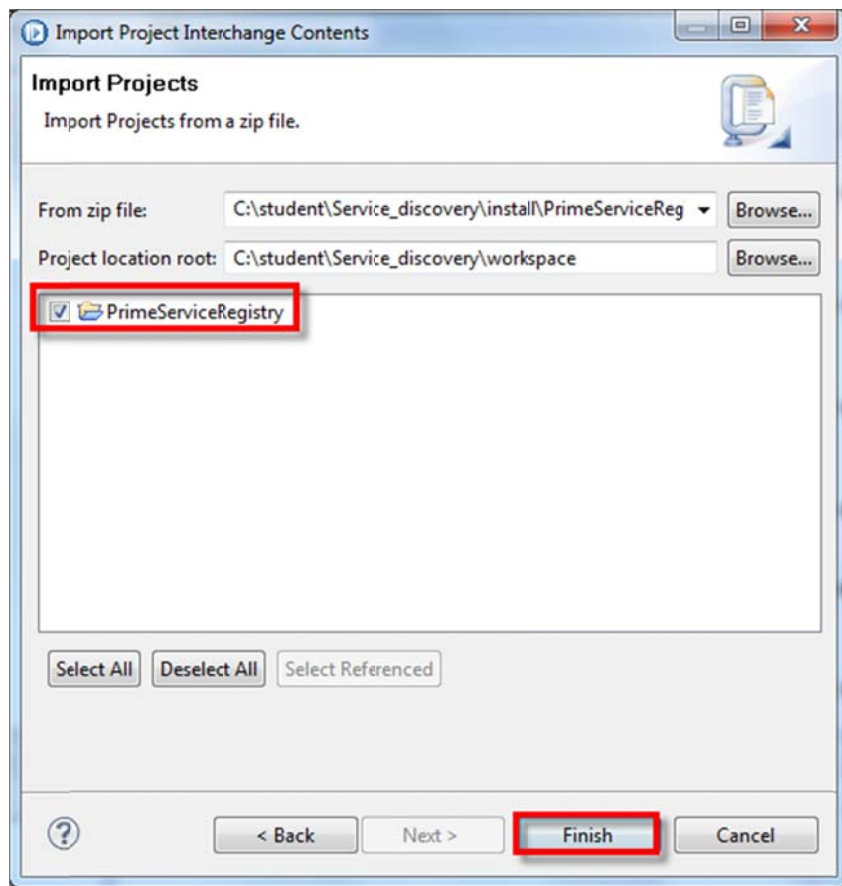
- ___5. Click **Browse...** on the Import Project Interchange Contents window.



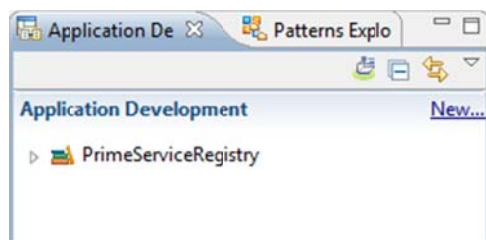
- ___6. Change to the **c:\student\service_discovery\install** directory, and select the **PrimeServiceRegistry** file. Click **Open**.



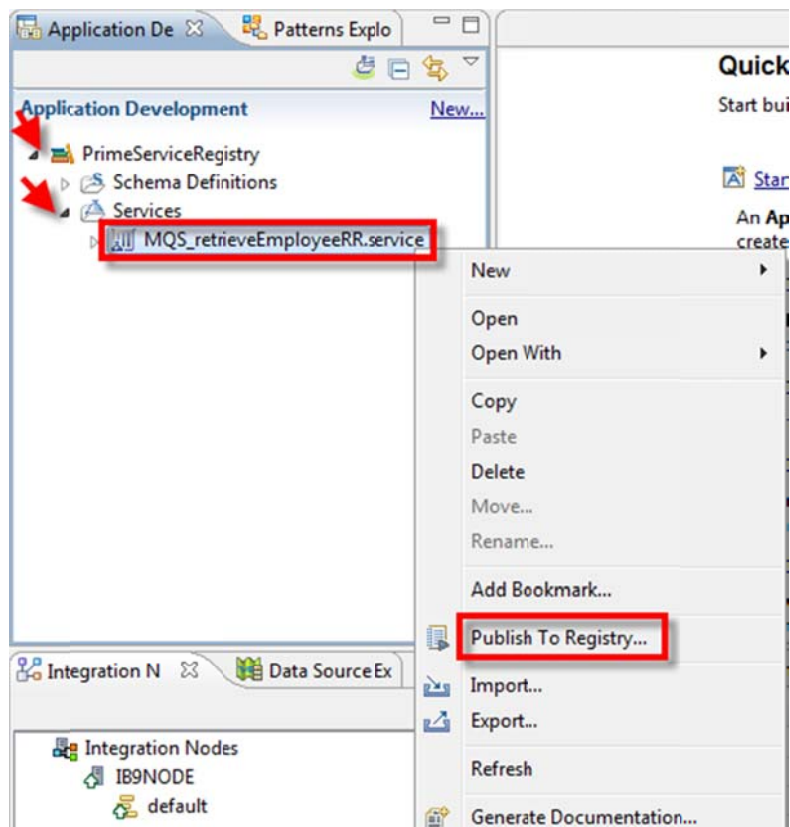
- ___7. Ensure the box is checked next to **PrimeServiceRegistry** then click **Finish**.



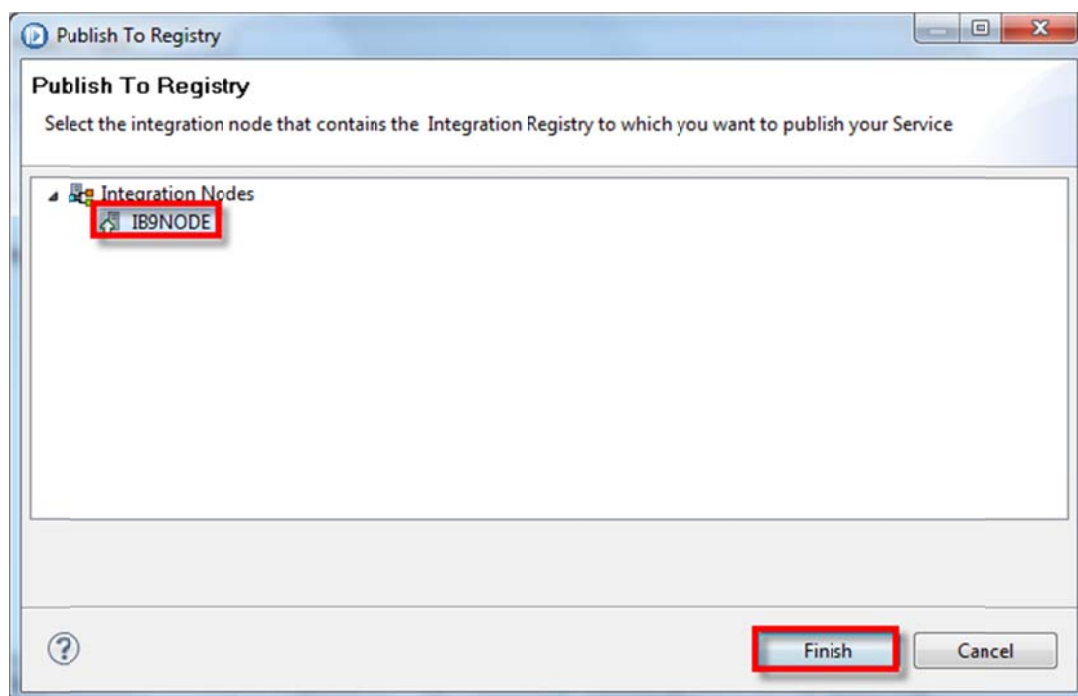
- ___8. The project gets imported, and a Library with the same name as the PI file will be created.



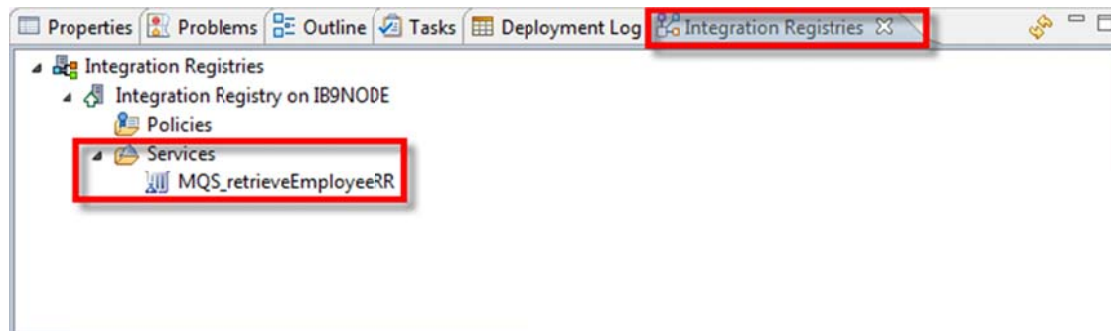
- ___9. Expand the Library until you see the MQ Service **MQS_retrieveEmployeeRR.service**. Right-click on this service definition and select **Publish to Registry...** from the menu.



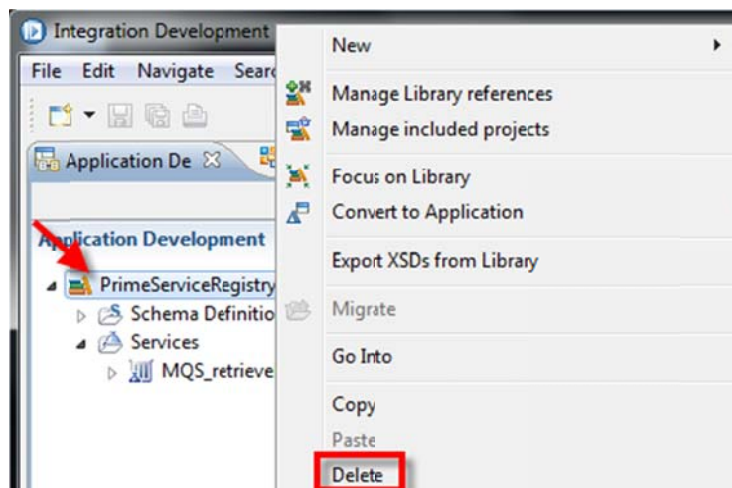
- ___10. Select the **IB9Node** and click **Finish**.



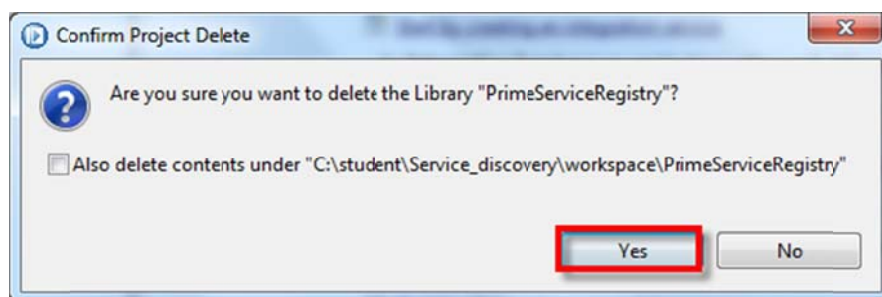
- ___11. After a few seconds the Integration Registry view will open automatically, and the MQ service is stored in the Services folder.



- ___12. Now delete the **PrimeServiceRegistry** library. Right-click on the **PrimeServiceRegistry** Library and select **Delete**.



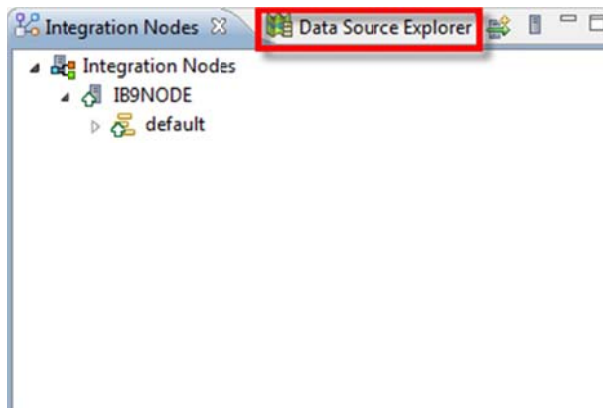
Select **Yes** to confirm the deletion.



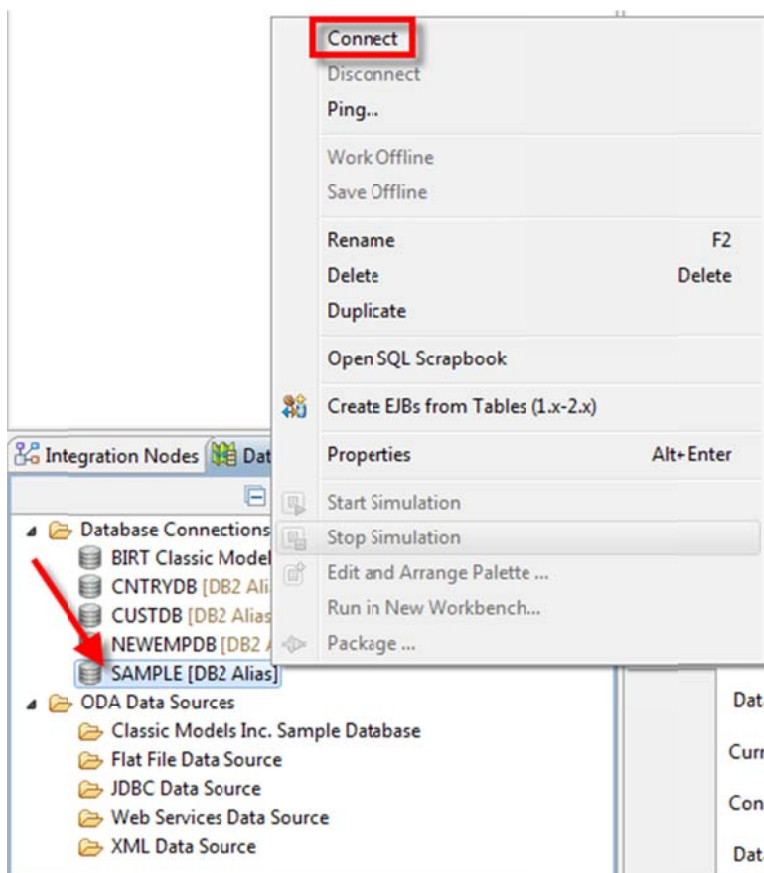
5.2.2 Building the SAMPLE connection

The database connection for SAMPLE already exists. We will connect to it so we can use it.

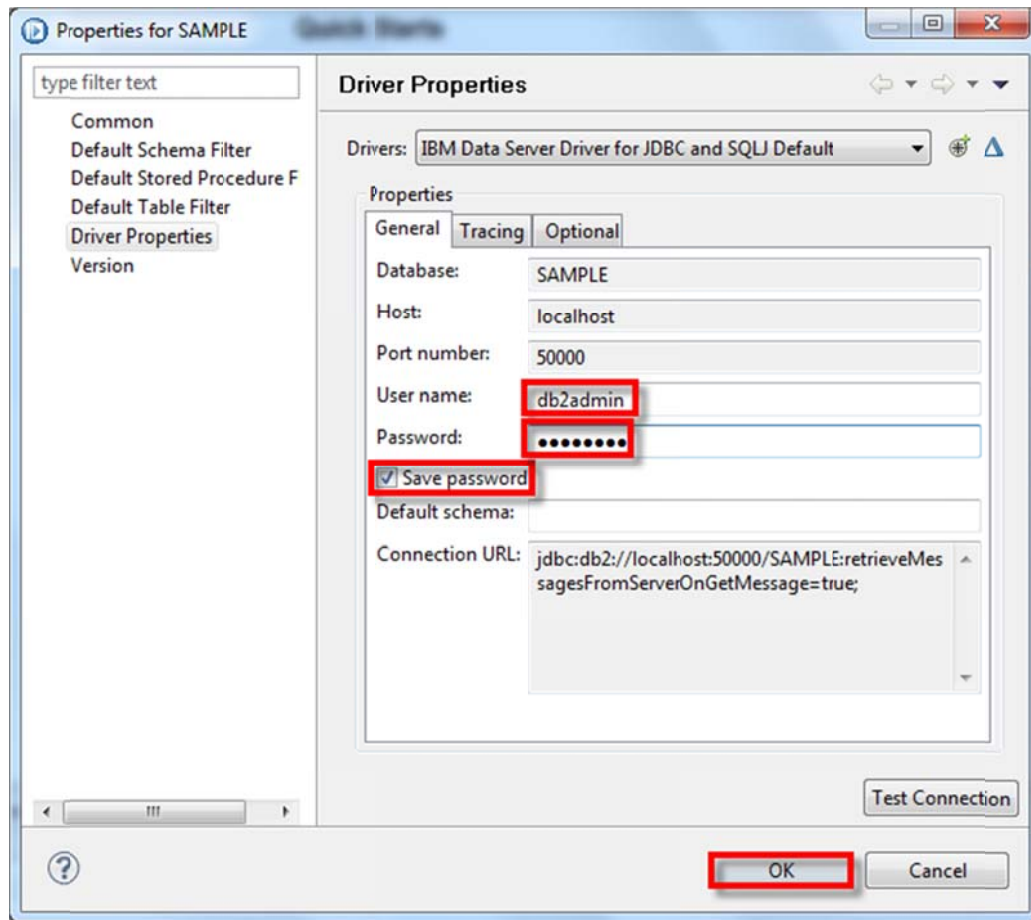
1. Click on the **Data Source Explorer** view to open it.



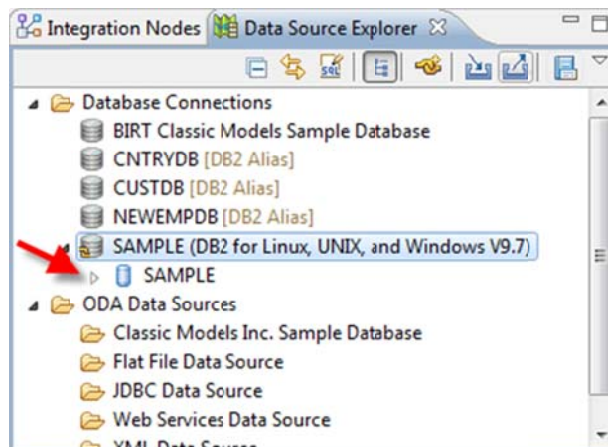
2. Right-click the **SAMPLE** Database Connection, and select **Connect** from the menu.



- ___3. Set the User name to **db2admin** and Password to **db8admin**. Also click the **Save** password checkbox. Click **OK**.



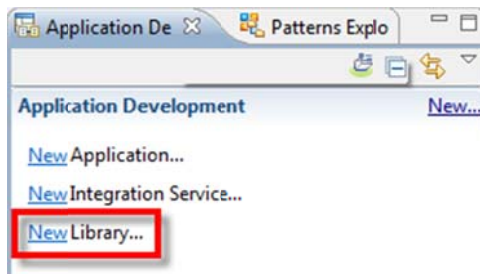
- ___4. A JDBC connection will be established to the SAMPLE database, and you will see the **SAMPLE** database with a blue “connected” database icon.



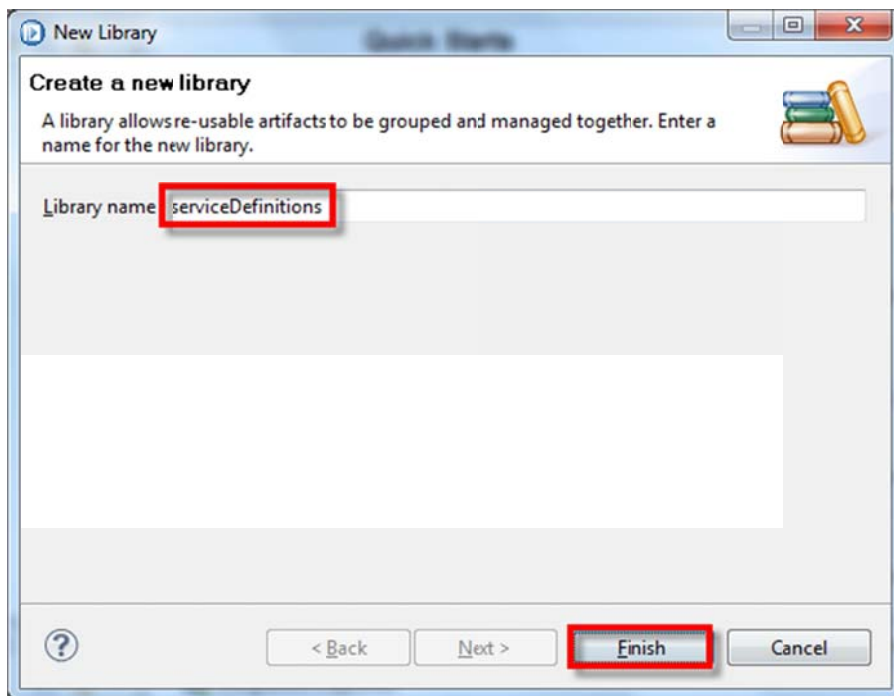
We can now build our database definition file.

- __5. Create a new library to hold the service definitions.

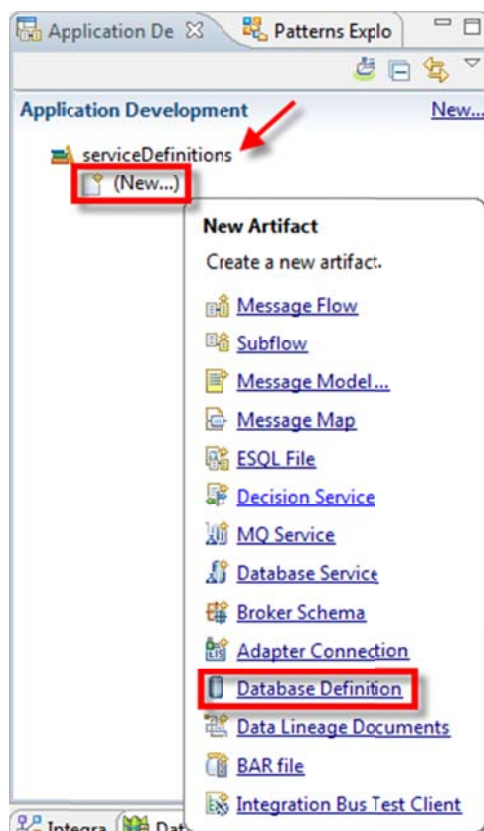
In the Application Development explorer, click **New Library....**



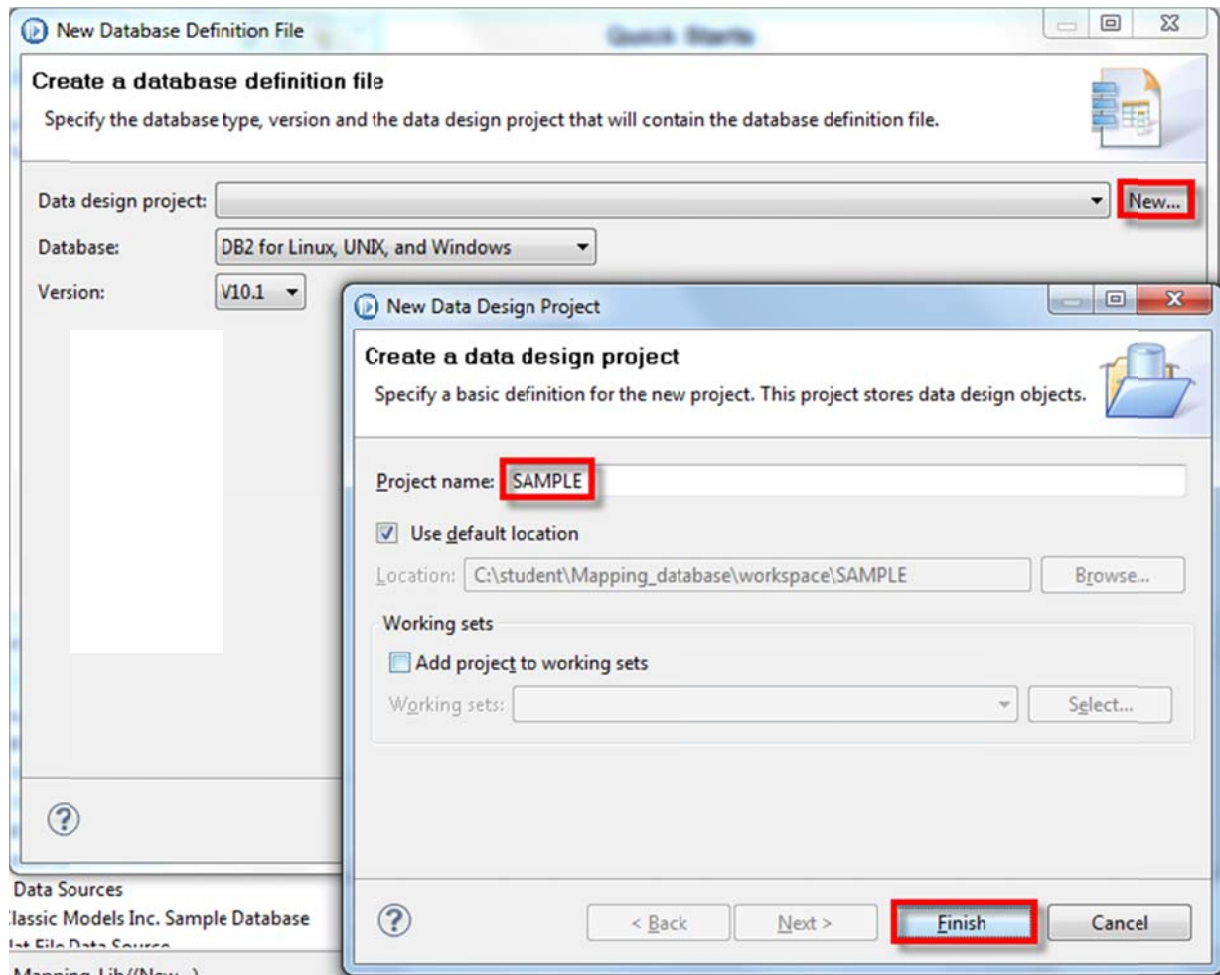
- __6. Name the library **serviceDefinitions** and click **Finish**.



___7. In the **serviceDefinitions** library, click **New**, and select **Database Definition**.

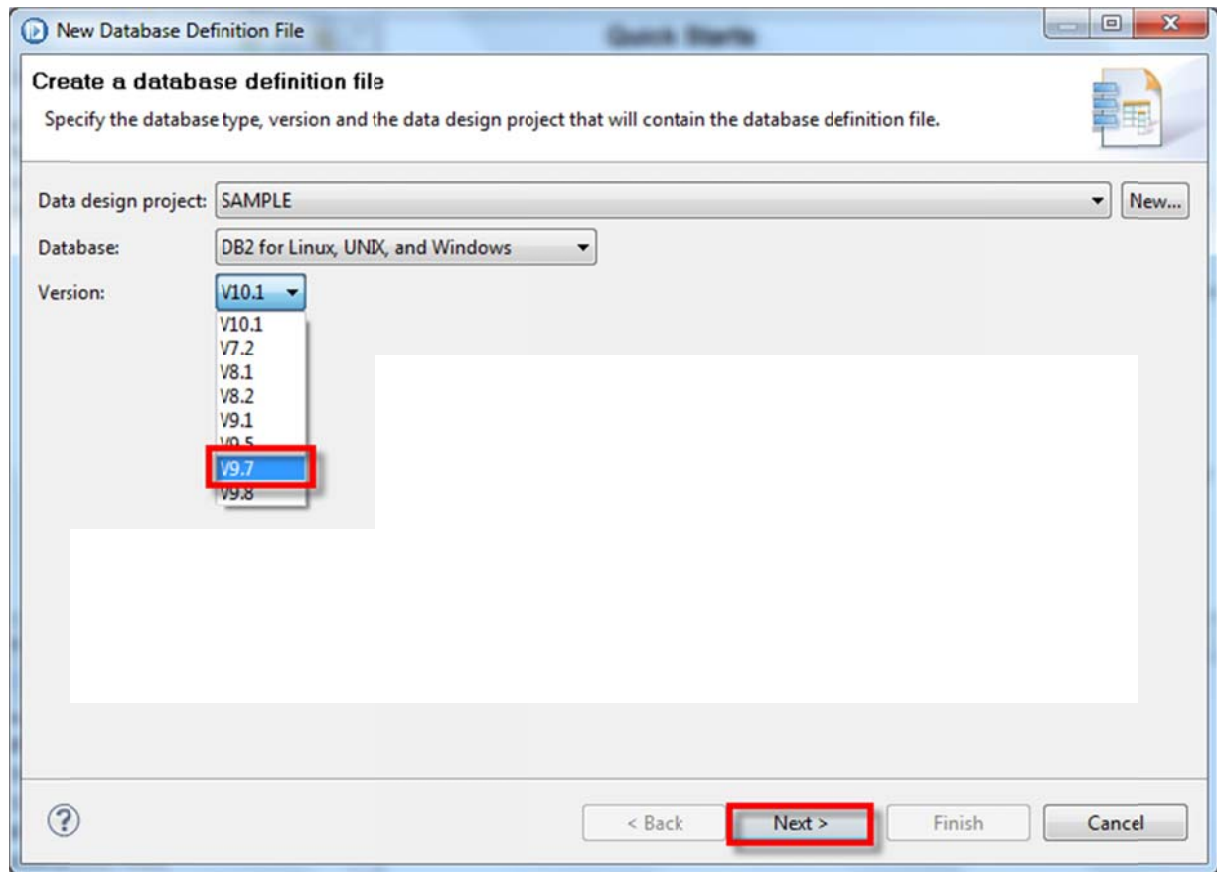


- ___8. Since we do not have an existing project to store this, click **New** to create a new database design project. Name this **SAMPLE**.



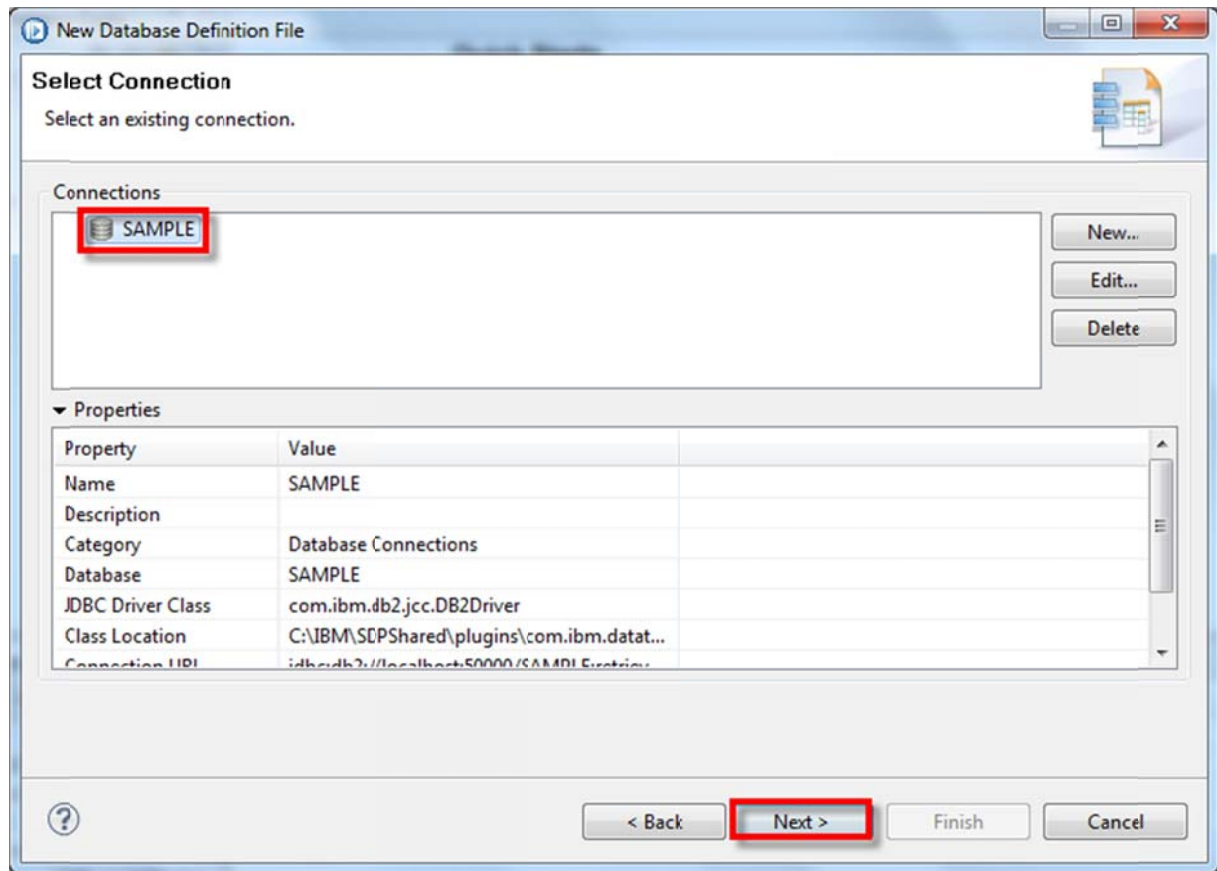
- ___9. Click **Finish**.

- ___10. The Database Design project field has now been populated with the SAMPLE project. Select **V9.7** from the **Version** drop-down box.

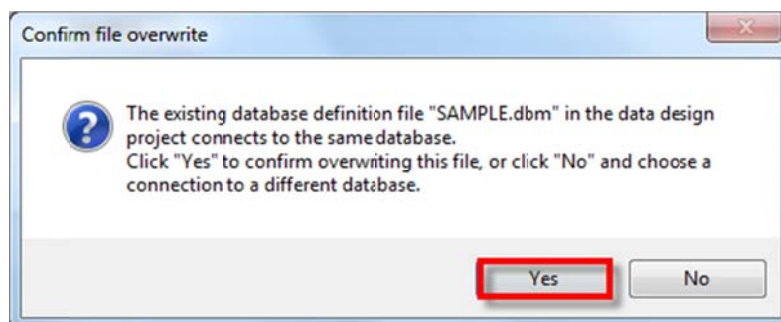


- ___11. Click **Next**.

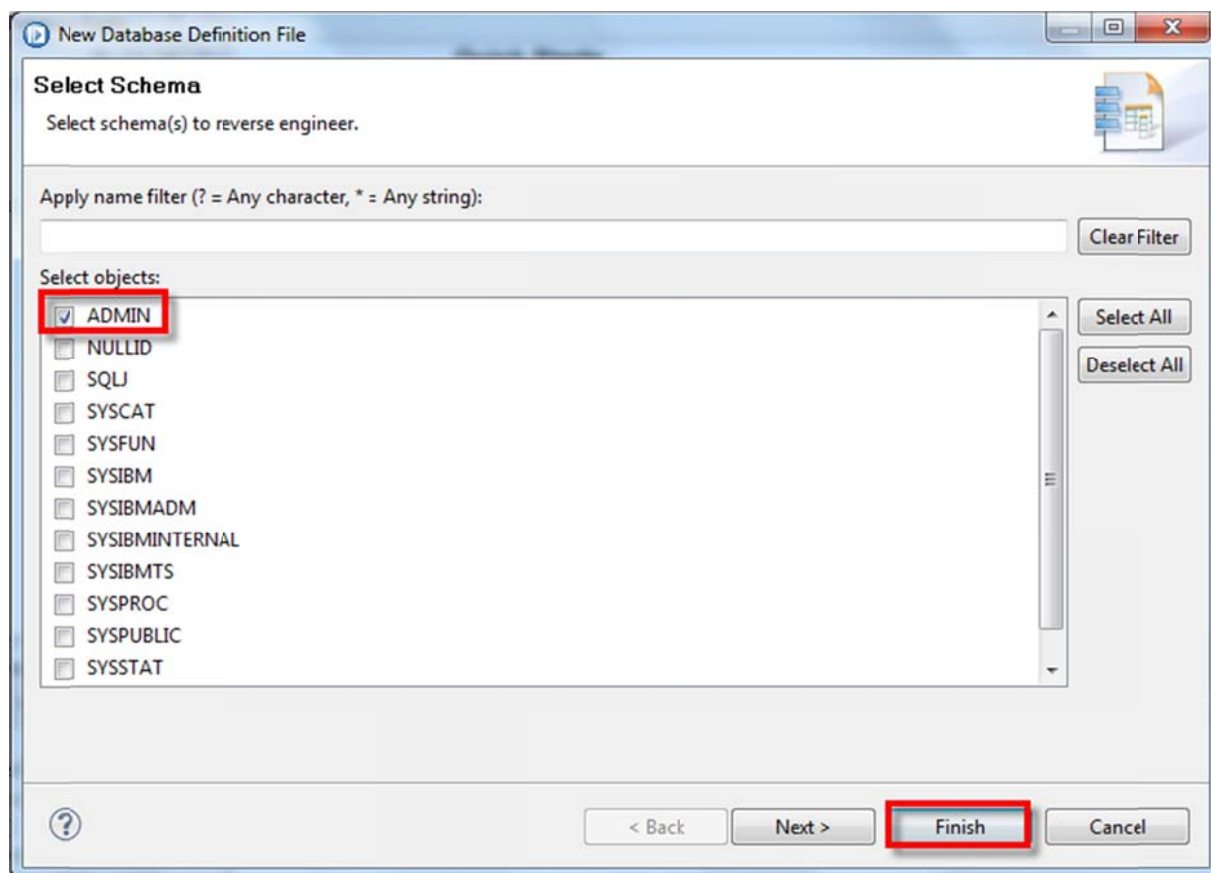
__12. Highlight the **SAMPLE** connection that we have connected to and click **Next**.



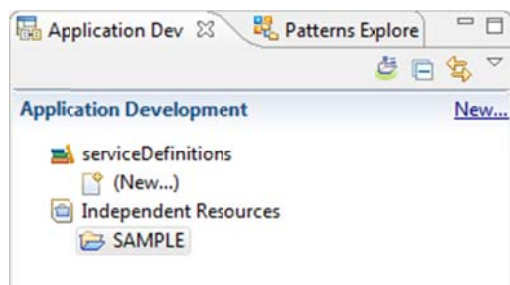
__13. If you get a **Confirm file overwrite** confirmation, click **Yes**.



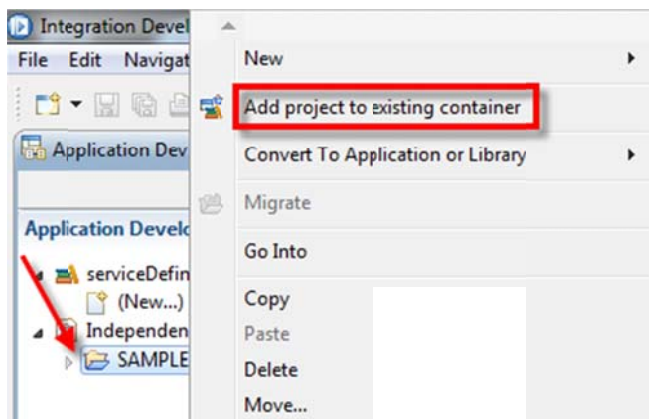
- ___14. Select the **ADMIN** schema (the SAMPLE database was created under the ADMIN schema). Click **Finish**.



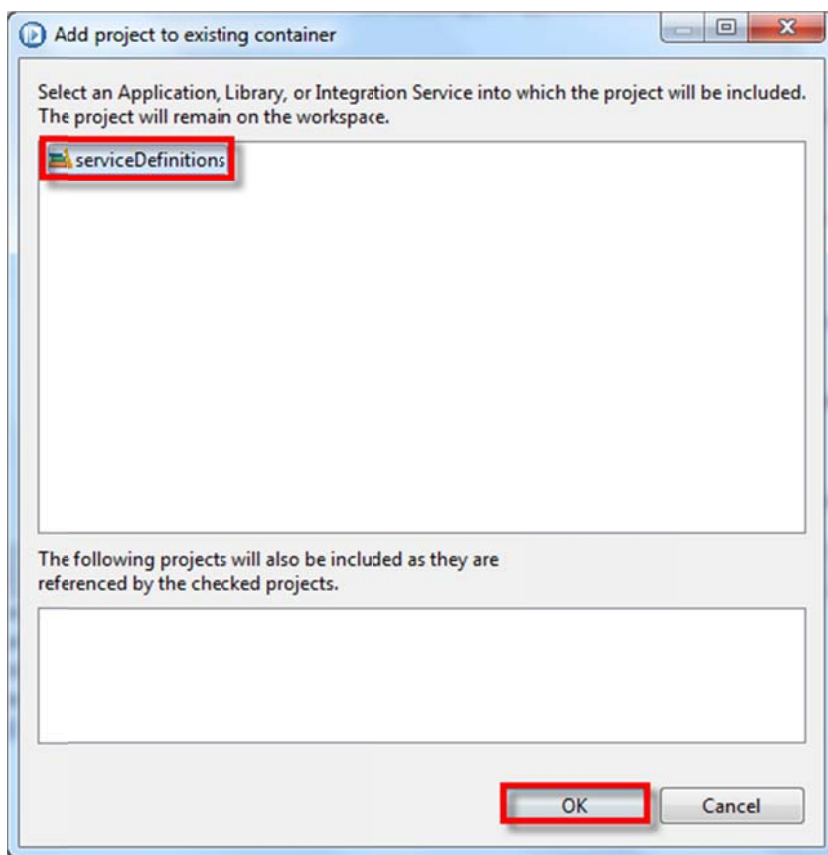
- ___15. The Database Design project has now been created under the Independent Resources folder.



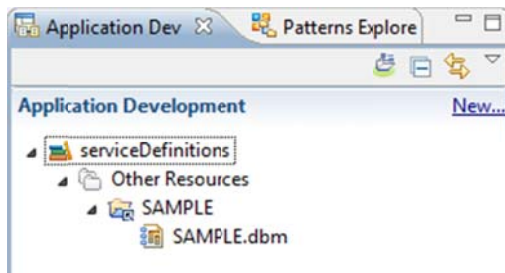
- ___16. Right-click on the **SAMPLE** Database Design project, and select **Add project to existing container** from the menu.



- ___17. Click on the **serviceDefinitions** library, and then click **OK**.



- ___18. The Database Definition file will now be in the serviceDefinitions library. It is now available to the application and does not need to be changed further. If it is open, close the **SAMPLE.dbm** editor.

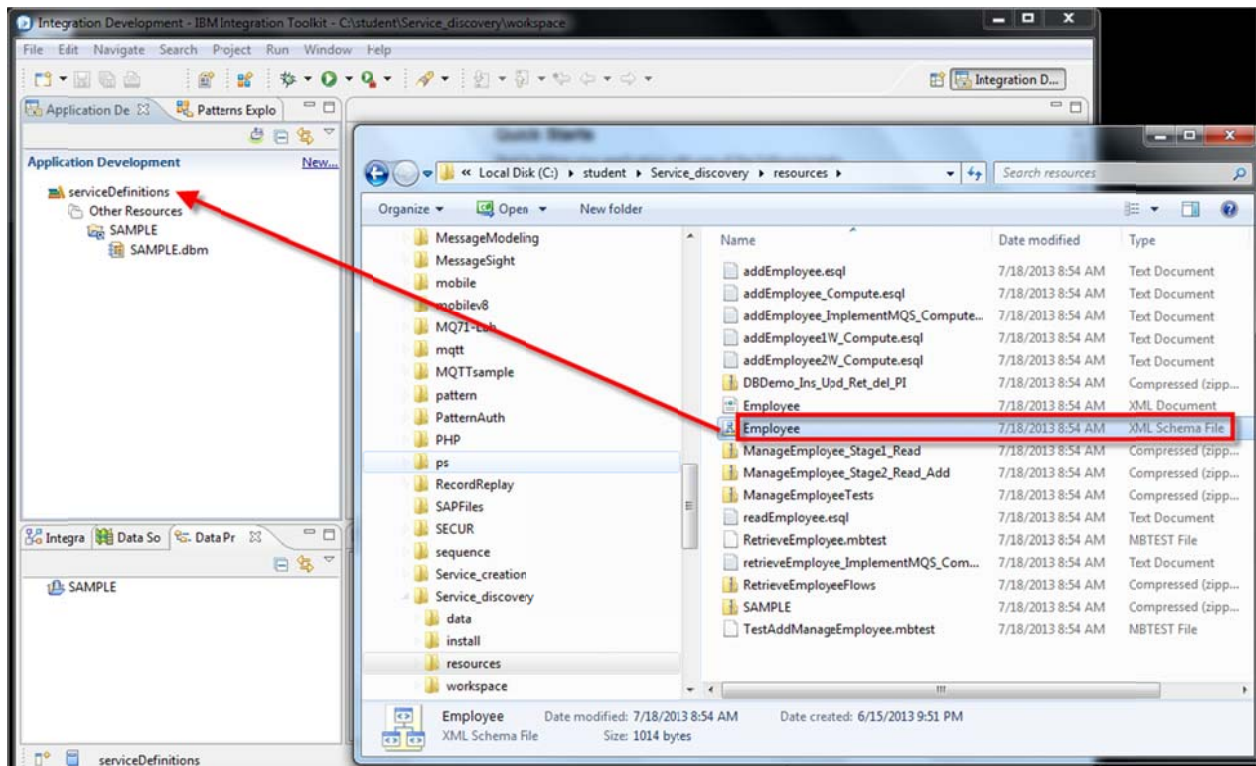


5.2.3 Adding a schema and creating MQ queues

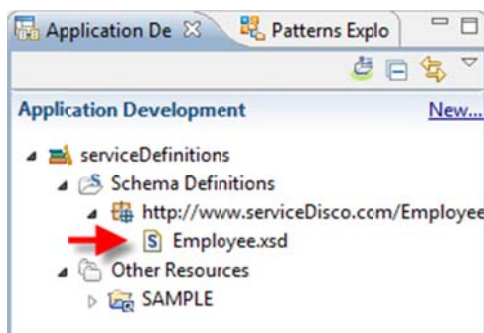
- ___1. Start Windows Explorer if not already running, by clicking on the Windows Explorer icon in the task bar.



- ___2. In Windows Explorer, navigate to **C:\student\Service_discovery\resources** and drag the **Employee.xsd** file on to the Integration Bus Toolkit and drop the file on to the **serviceDefinitions** Library.



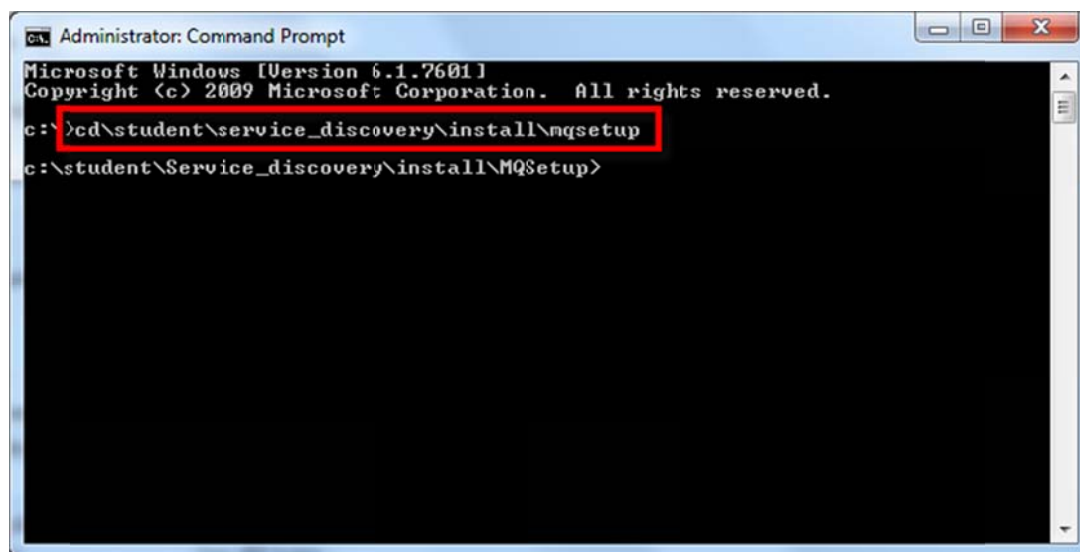
The schema will be shown in the **serviceDefinitions** project. We will use this when creating the MQ Service definitions to define the message types used by the MQ service.



- ___3. We now need to create the MQ queues used by the lab. There is a mqsc script file available to create the queues. Open a Command Prompt by clicking on the Command Prompt icon in the task bar.

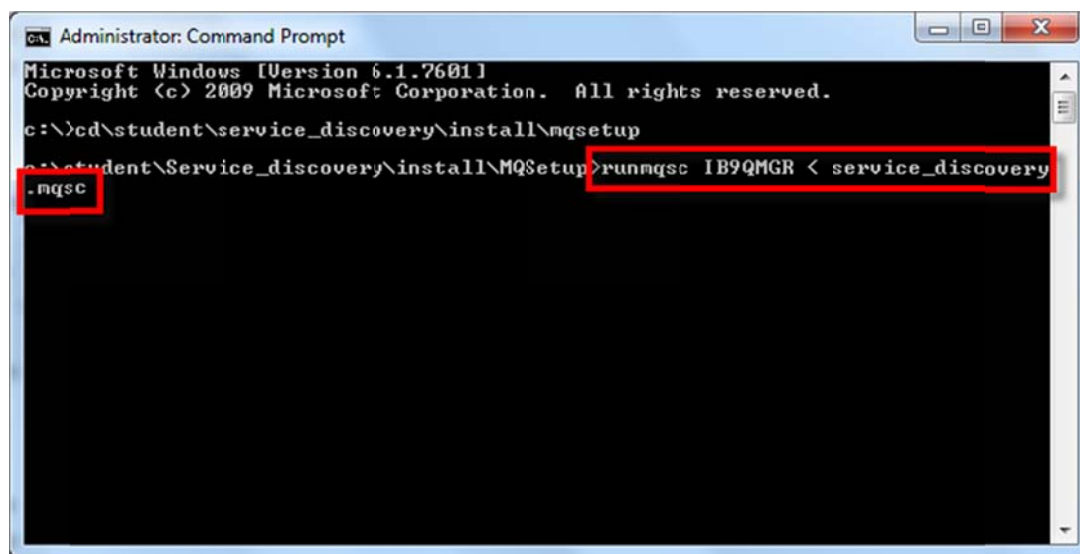


- ___4. In the Command Prompt, enter **cd\student\service_discovery\install\mqsetup**



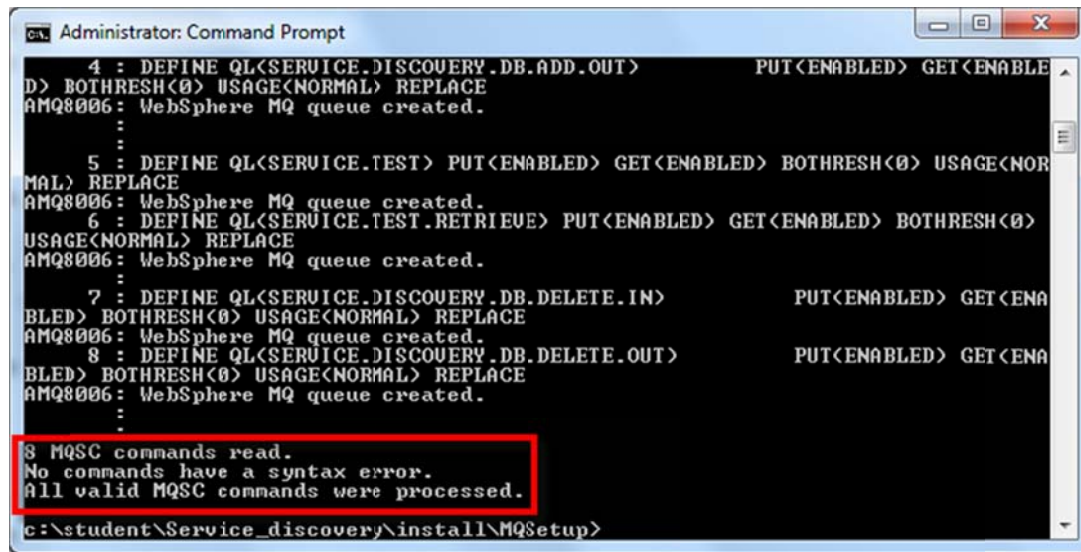
A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window shows the following text: "Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved." followed by the command "c:\>cd\student\service_discovery\install\mqsetup" which is highlighted with a red box. The prompt then shows "c:\student\Service_discovery\install\MQSetup>" on the next line.

- ___5. In the Command Prompt, enter **runmqsc IB9QMGR < service_discovery.mqsc**



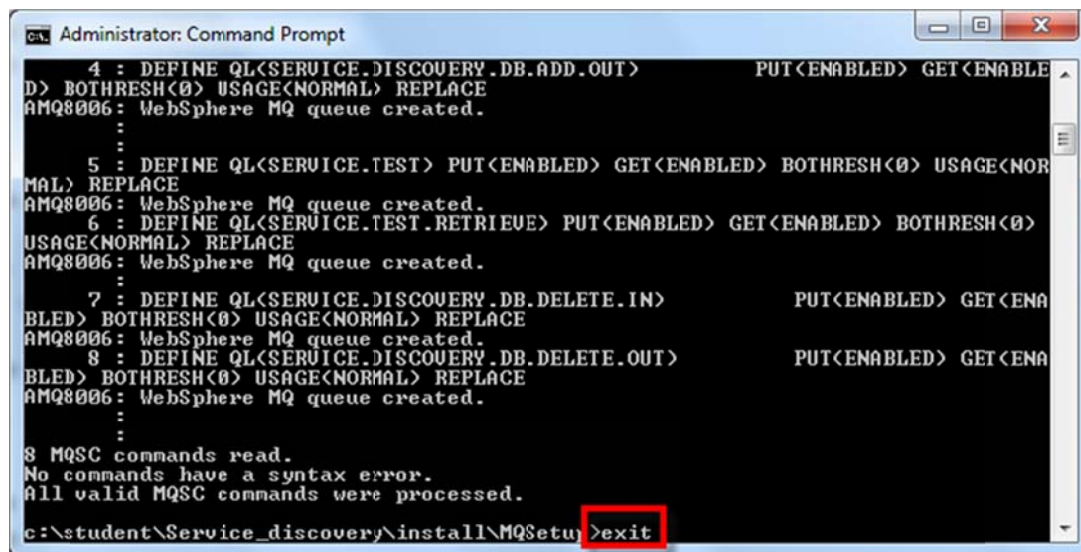
A screenshot of a Windows Command Prompt window titled "Administrator: Command Prompt". The window shows the following text: "Microsoft Windows [Version 6.1.7601] Copyright (c) 2009 Microsoft Corporation. All rights reserved." followed by the command "c:\>cd\student\service_discovery\install\mqsetup". On the next line, the command "c:\student\Service_discovery\install\MQSetup>runmqsc IB9QMGR < service_discovery.mqsc" is shown, with "runmqsc IB9QMGR < service_discovery" highlighted by a red box and ".mqsc" highlighted by another red box on the line below.

- ___6. The script will execute, and you should see that there were no errors.



```
Administrator: Command Prompt
4 : DEFINE QL(SERVICE.DISCOVERY.DB.ADD.OUT) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
:
5 : DEFINE QL(SERVICE.TEST) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
6 : DEFINE QL(SERVICE.TEST.RETRIEVE) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
:
7 : DEFINE QL(SERVICE.DISCOVERY.DB.DELETE.IN) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
8 : DEFINE QL(SERVICE.DISCOVERY.DB.DELETE.OUT) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
:
8 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
c:\student\Service_discovery\install\MQSetup>
```

- ___7. In the Command Prompt, enter **exit** to close the window.



```
Administrator: Command Prompt
4 : DEFINE QL(SERVICE.DISCOVERY.DB.ADD.OUT) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
:
5 : DEFINE QL(SERVICE.TEST) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
6 : DEFINE QL(SERVICE.TEST.RETRIEVE) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
:
7 : DEFINE QL(SERVICE.DISCOVERY.DB.DELETE.IN) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
8 : DEFINE QL(SERVICE.DISCOVERY.DB.DELETE.OUT) PUT(ENABLED) GET(ENABLED) BOTHRESH(0) USAGE(NORMAL) REPLACE
AMQ8006: WebSphere MQ queue created.
:
8 MQSC commands read.
No commands have a syntax error.
All valid MQSC commands were processed.
c:\student\Service_discovery\install\MQSetup>exit
```

5.3 MQ Services

5.3.1 Background

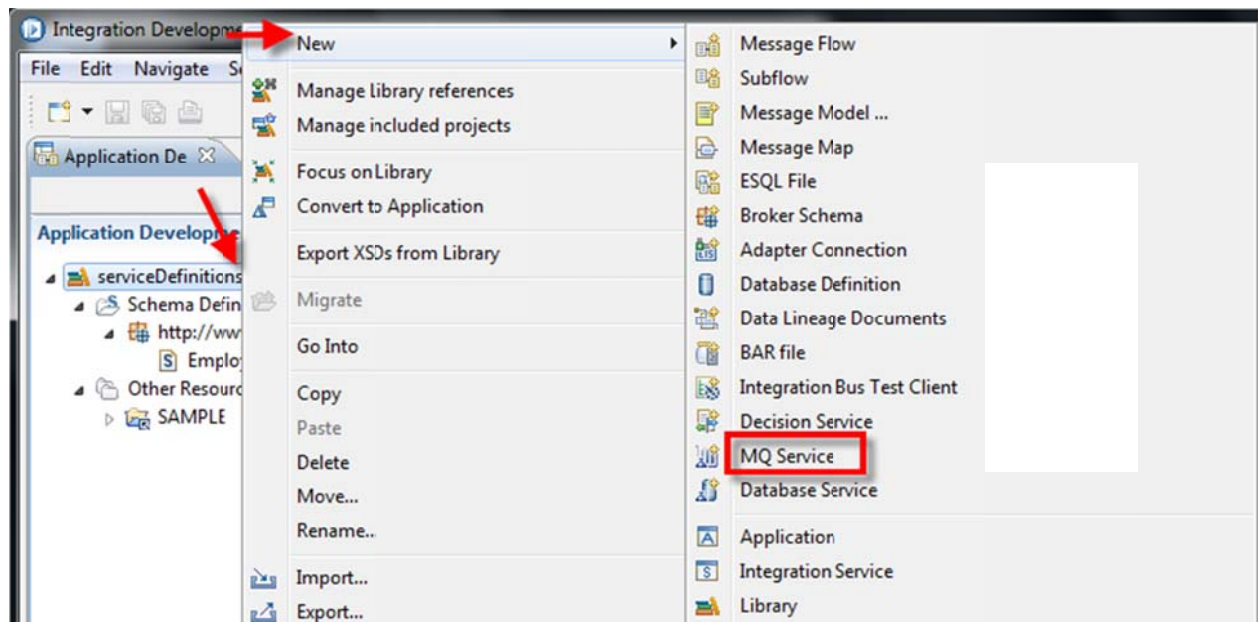
Think of an MQ Service as an MQ application that provides information via one or more MQ resources. For the purposes of this lab guide, you will create a “Request-Response” MQ service that will be used to develop an application called ManageEmployee. The application will also use a discovered database service to manage and retrieve data in a database.

Once discovered, MQ services can be stored in the Integration Registry component allowing other developers to develop applications using the discovered services. You will also import an MQ Service that has been developed by another Integration Bus developer and use this service to extend the ManageEmployee application.

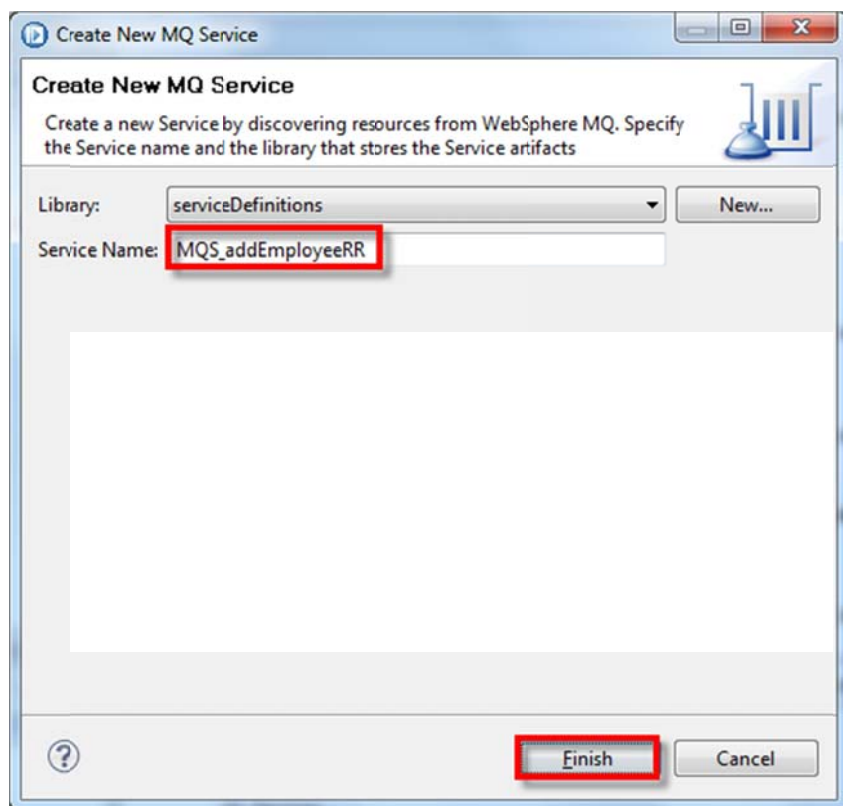
5.3.2 Create a MQ request response service definition

The following sections will guide you through creating MQ Service definitions for Queues that will be used later in this lab guide.

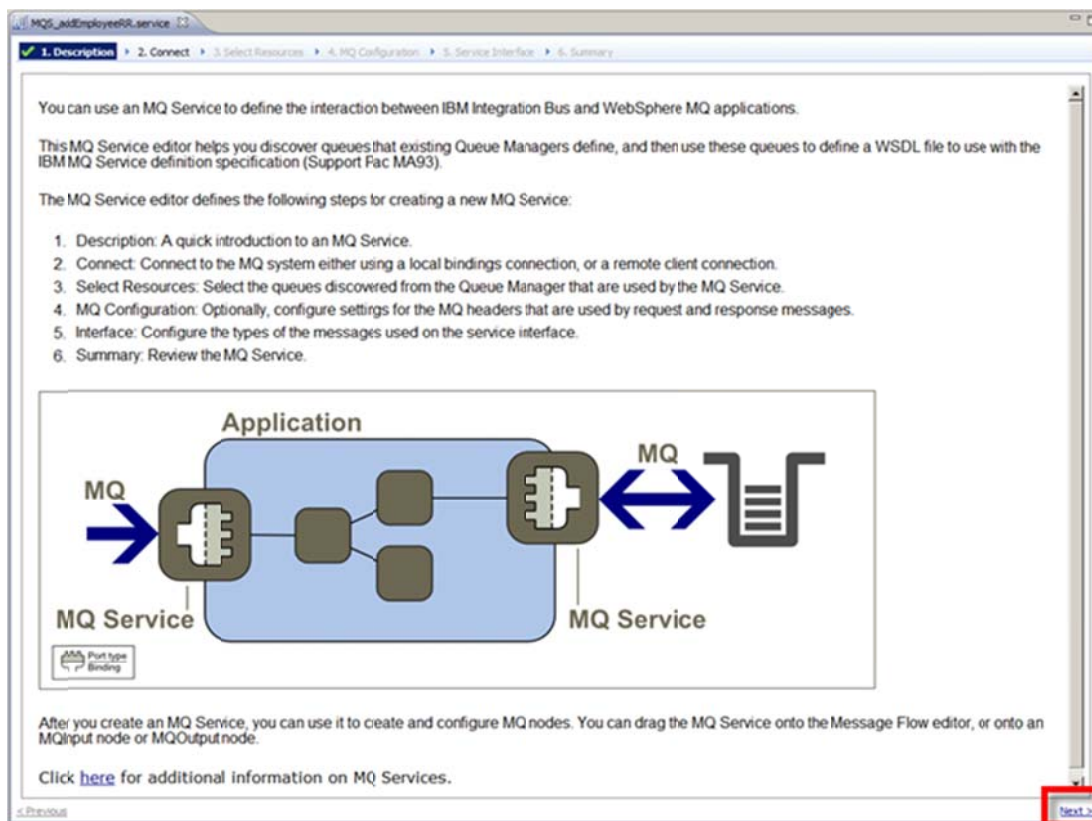
1. Right-click on the **serviceDefinitions** Library and choose **New**→**MQ Service**.



- ___2. In the Create New MQ Service window, set the **Service Name** to **MQS_addEmployeeRR** and click **Finish**.



- __3. The MQ Service creation editor will open. Click **Next** after reading the description to move to the next step.



- ___4. Specify the queue manager name as **IB9QMGR** and press the **Test Connection** link to ensure that the connection attempt was successful. If successful, you get the **Connection was tested successfully** feedback.

The screenshot shows a web-based configuration interface for a service named "MQS_addEmployeeRR.service". The interface has a breadcrumb trail at the top: 1. Description > 2. Connect > 3. Select Resources > 4. MQ Configuration > 5. Service Interface > 6. Summary. The current step is "2. Connect".

Below the breadcrumb trail, there is a section titled "Specify connection parameters for the remote system to discover resources". Inside this section, a message box states "i Connection was tested successfully". Below this message, there are several input fields:

- Type of connection: Local binding connection (dropdown menu)
- Queue Manager Name: IB9QMGR (text field, highlighted with a red box)
- CCDT file URL: (text field with a "Select..." button)
- Host or IP: localhost (text field)
- Port: 1414 (text field)
- Channel Name: SYSTEM.BKR.CONFIG (text field)

At the bottom of the form, there is a "Test Connection" button, which is also highlighted with a red box. At the very bottom of the window, there are links for "< Previous" and "Next >".

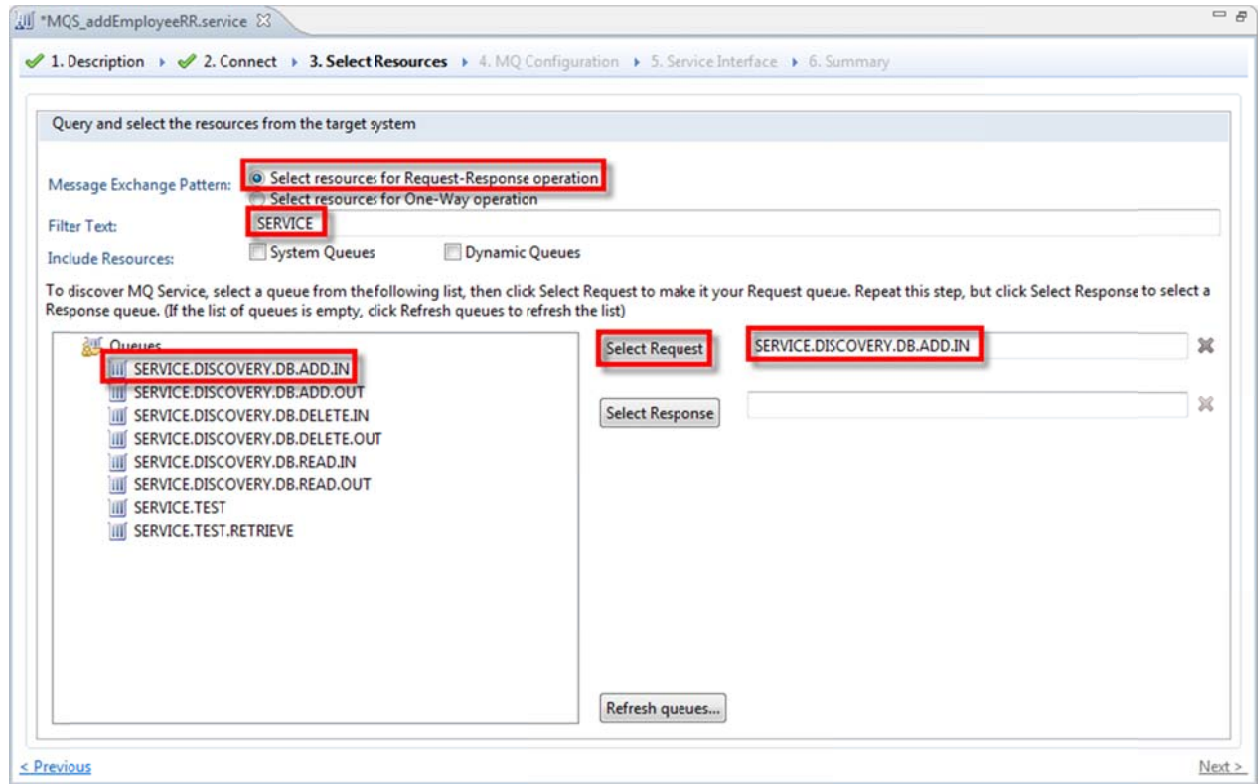
- ___5. Click **Next**.

__6. The dialog will show you a list of queues from the IB9QMGR queue manager.

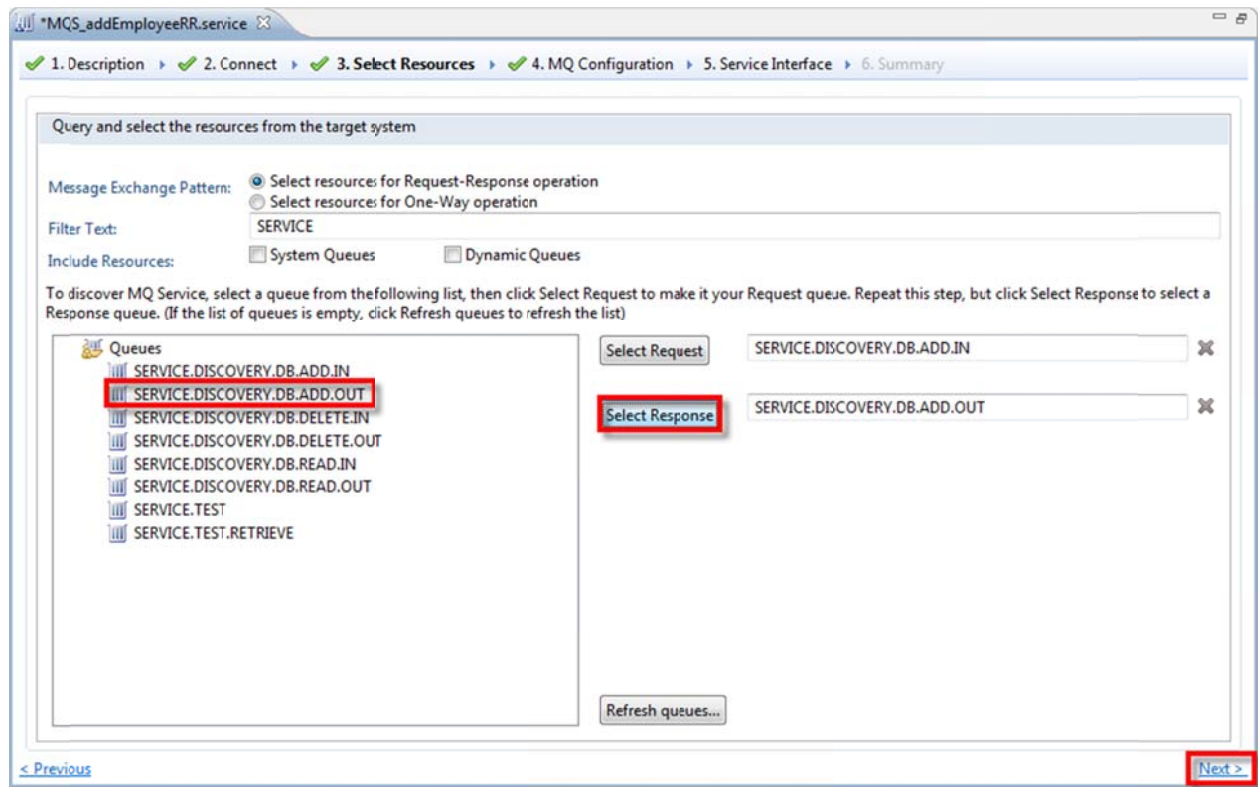
Choose **Select Resources for Request-Response operation**.

To restrict the number of queues in the view, in the **Filter Text** field type **SERVICE** (this list of queues with SERVICE in the first part of the name will appear).

Highlight **SERVICE.DISCOVERY.DB.ADD.IN** in the list of queues and click the **Select Request** button (the queue name will be copied to the Select Request field).



- ___7. Now highlight **SERVICE.DISCOVERY.DB.ADD.OUT** in the list of queues and click the **Select Response** button (the queue name will be copied to the Select Response field).



- ___8. Click **Next**.

- ___9. Note that Request / Response message headers can be configured. We do not need changes, so click **Next**.

The screenshot shows the 'MQ Configuration' step of a wizard for the service '*MQS_addEmployeeRR.service'. The progress bar at the top indicates steps 1 through 6, with step 4, 'MQ Configuration', currently selected. The main panel is titled 'Configure MQ headers for request and response messages'. It contains two sections: 'Request Message Headers' and 'Response Message Headers'. The 'Request Message Headers' section includes fields for CCSID, Format, Message type (set to 'Request'), Persistence, Message ID, Correlation ID, Expiry, and Priority. The 'Response Message Headers' section includes fields for CCSID and Format. At the bottom left is a '< Previous' link, and at the bottom right is a 'Next >' button, which is highlighted with a red rectangle.

- ___10. On the “5. Service Interface window,” click the **Click to select message...** link for the **Input** Operation.

1. Description 2. Connect 3. Select Resources 4. MQ Configuration 5. Service Interface 6. Summary

Configure your interface and advanced binding parameters

Interface

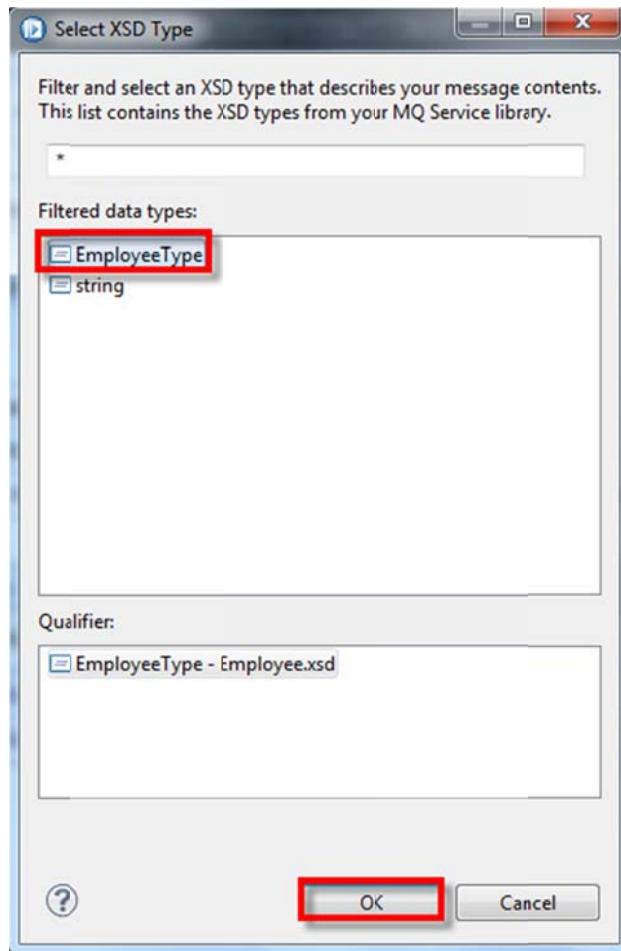
Name	MQS_addEmployeeRR
Namespace	http://MQS_addEmployeeRR
Message Configuration	<input checked="" type="radio"/> Use XML schema types <input type="radio"/> Use XML schema elements

Operations

Operation	Message
SERVICEDISCOVER/DBADDIN_SERVICEDI/COVEYDBADDOUT_Operation	
Input	Click to select message... Open file...
Output	Click to select message... Open file...

< Previous Next >

- ___11. In the Select XSD Type window, highlight **EmployeeType** from the list of Filtered data types and click **OK**.



- ___12. Repeat steps 10 and 11 for the **Output** operation. The message type for the Input and Output operations will be updated with the EmployeeType.

MQS_addEmployeeRR.service

1. Description 2. Connect 3. Select Resources 4. MQ Configuration 5. Service Interface 6. Summary

Configure your interface and advanced binding parameters

Interface

Name	MQS_addEmployeeRR
Namespace	http://MQS_addEmployeeRR
Message Configuration	<input checked="" type="radio"/> Use XML schema types <input type="radio"/> Use XML schema elements

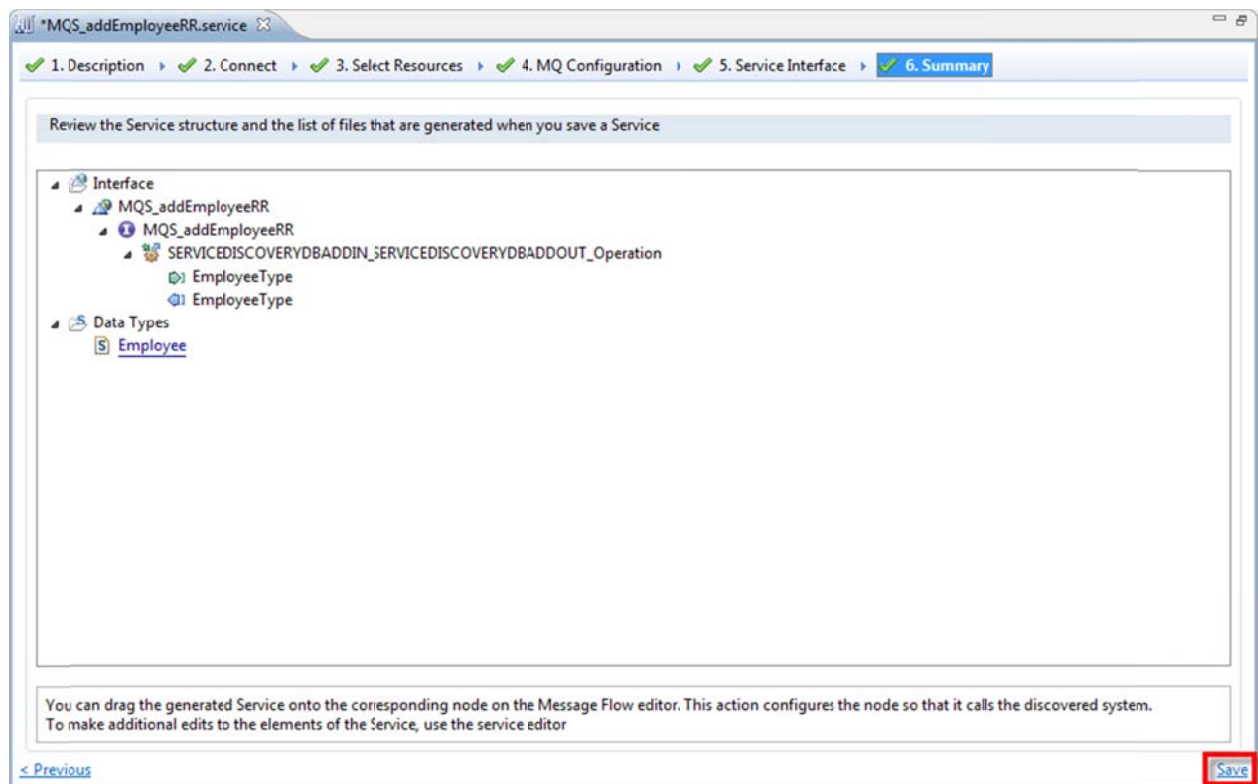
Operations

Operation	Message	
SERVICEDiscoveryBADDIN_SERVICEDiscoveryBADDOUT_Operation		
Input	SERVICEDiscoveryBADDIN_SERVICEDiscoveryBADDOUT_Operation [EmployeeType]	Open file...
Output	SERVICEDiscoveryBADDIN_SERVICEDiscoveryBADDOUT_OperationResponse [EmployeeType]	Open file...

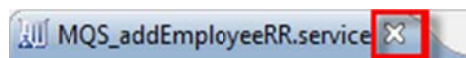
< Previous Next >

- ___13. Click **Next**.

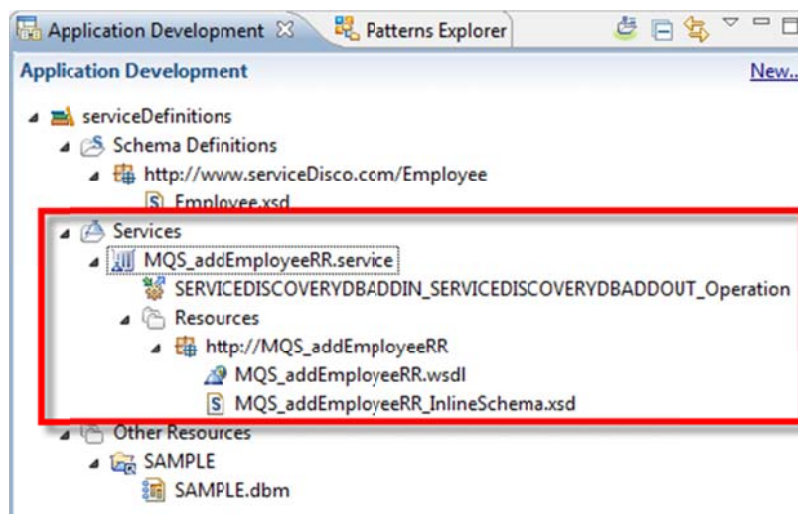
___14. On the final Summary screen, review the Service structure and click **Save**.



___15. Close the editor for the **MQS_addEmployeeRR** Request-Response MQ service.



___16. After it is saved, the Toolkit Navigator will show the newly created service under Services in the serviceDefinitions Library.



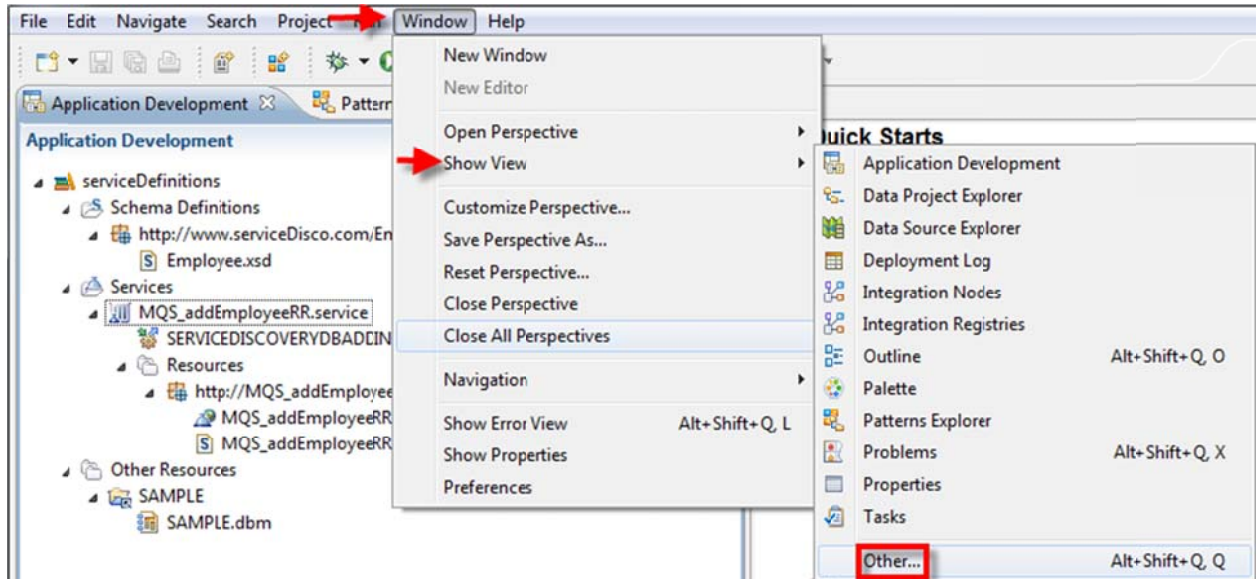
We will use this service when creating a message flow later in the lab guide.

5.3.3 Publish the MQ Service to an Integration Registry

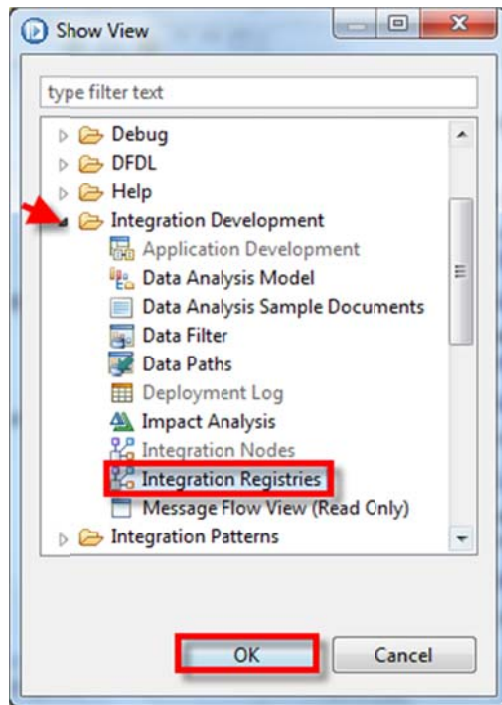
Discovered services can be stored in an Integration Registry. Each Integration Node has an Integration Registry that can be used to store Policies and Services. You will now store the MQ Service you just created into an Integration Registry associated with IB9NODE.

- ___1. Open the Integration Registries view in the IIB Toolkit. If it is already visible, skip ahead to step 3.

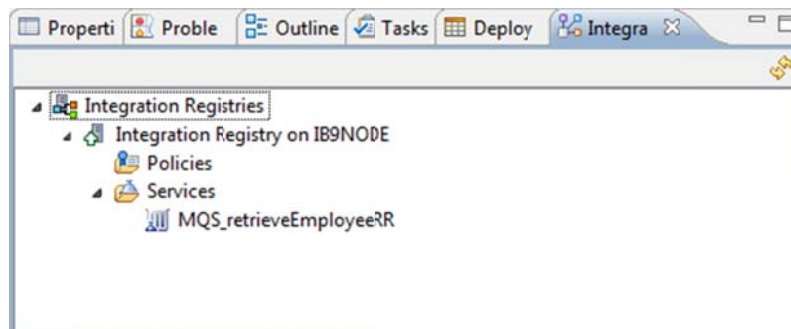
Select **Window**→**Show View**→**Other**



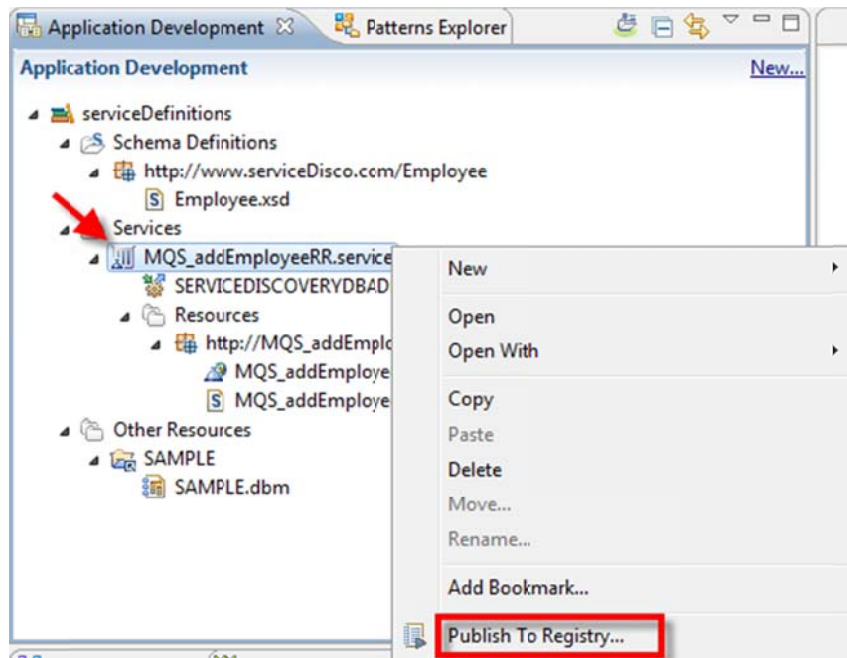
- __2. Open the **Integration Development** folder and select the **Integration Registries** view. Click **OK**.



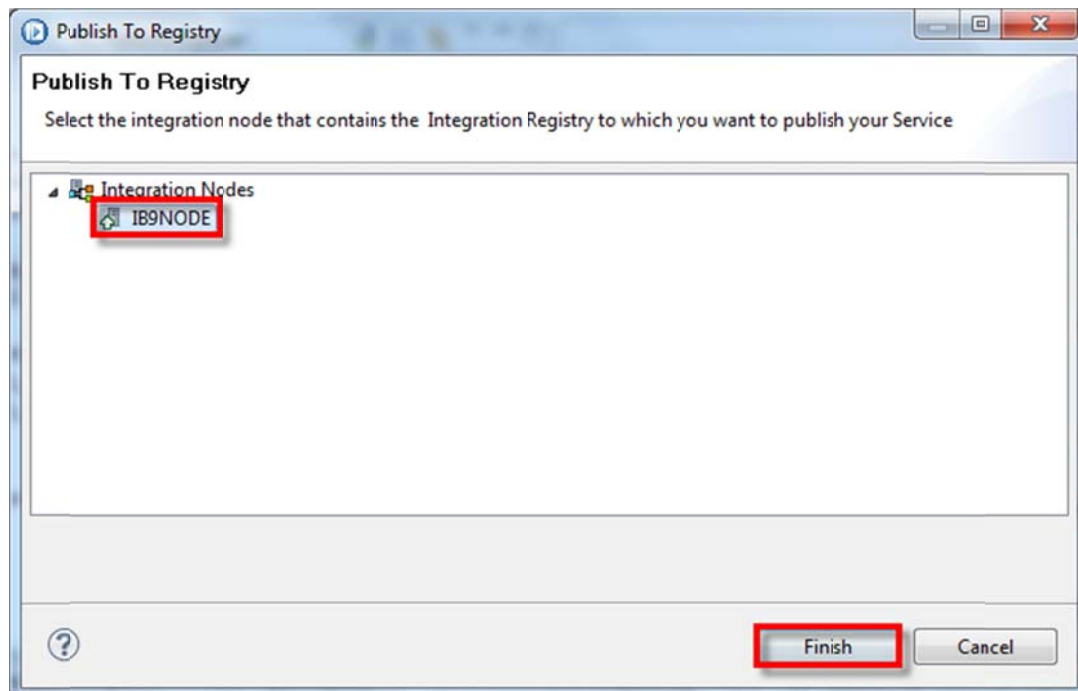
- __3. The view shows the contents of the **Integration Registry** associated with each IIB Node. The view will automatically attempt to connect to Integration nodes that are up and running and expand the IB9NODE Registry view to see two folders, Policies and Services.



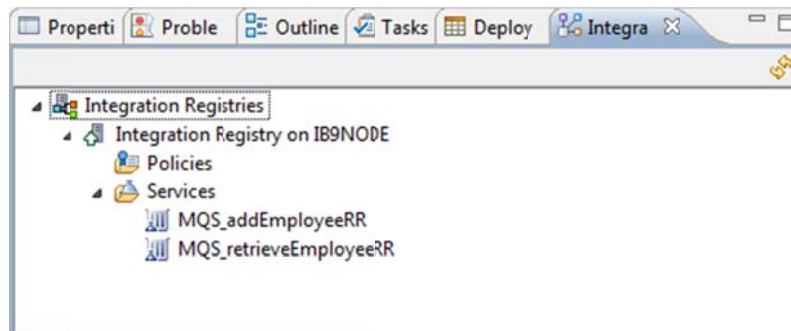
- ___4. In the Navigator view, right-click on the **MQS_addEmployeeRR.service** entry and click **Publish to Registry....**



- ___5. Select the **IB9NODE** on the dialog and click **Finish**.



- ___6. In the Integration Registry view, you will now see the MQS_addEmployeeRR service in the Services folder.



The service MQS_retrieveEmployeeRR has been stored in the registry by “another” Integration Bus developer and will be used later in this lab guide.

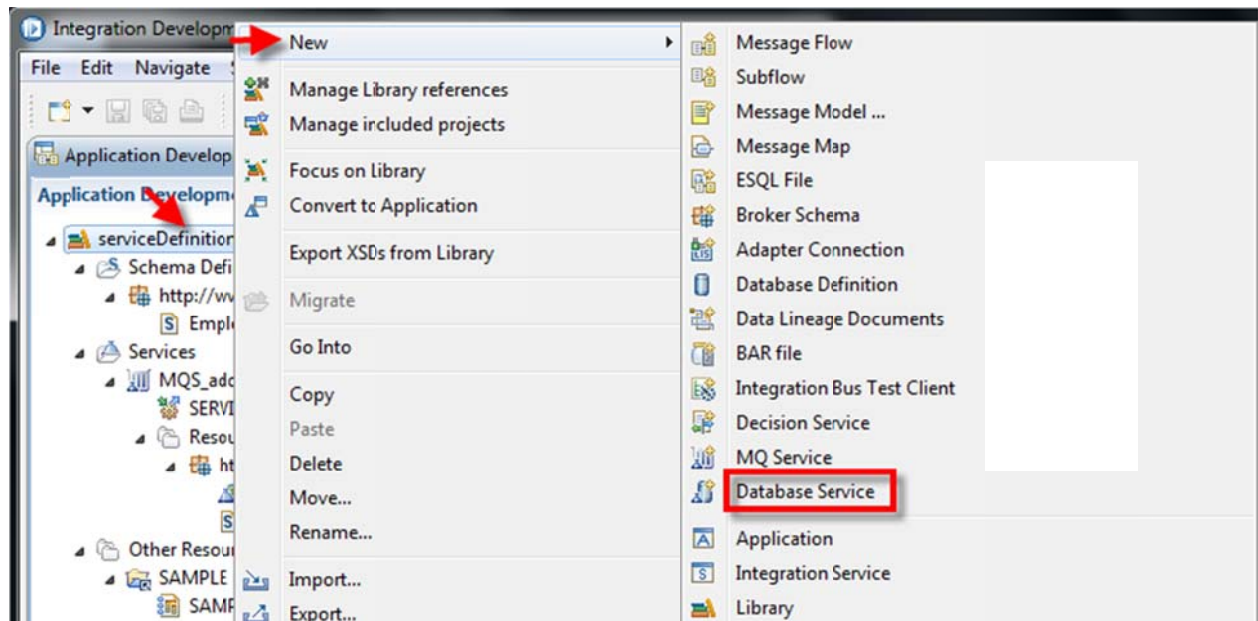
Other developers that have access to the IB9NODE Registry will be able to use this service definition stored in the IB9NODE Registry. Right-clicking on the service definition name and choosing the option **Import to workspace...** will copy the definition into a library of your choice within the workspace. (***There is no need to do this as we already have a copy of the service in the serviceDefinitions Library.***)

5.4 Database Services

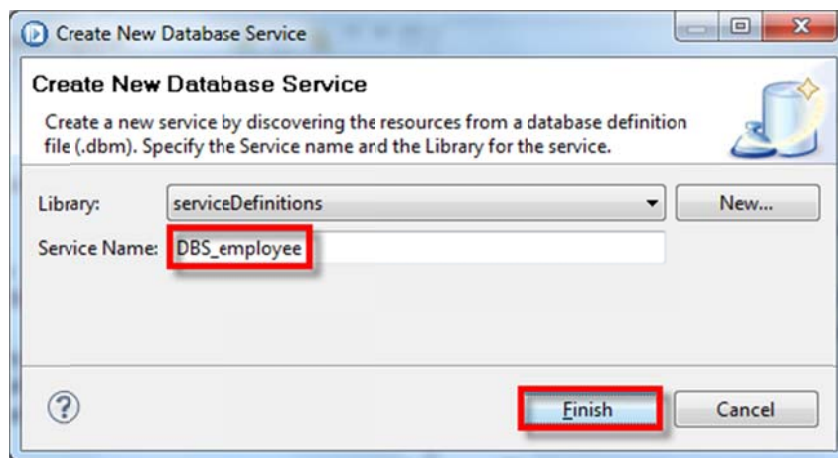
5.4.1 Discover the Database Service

You will now create a Database Service based on the SAMPLE dbm file that we imported earlier. You will create a Database Service that will be used to perform add (Insert) and retrieve (select) operations on an EMPLOYEE table in the SAMPLE database.

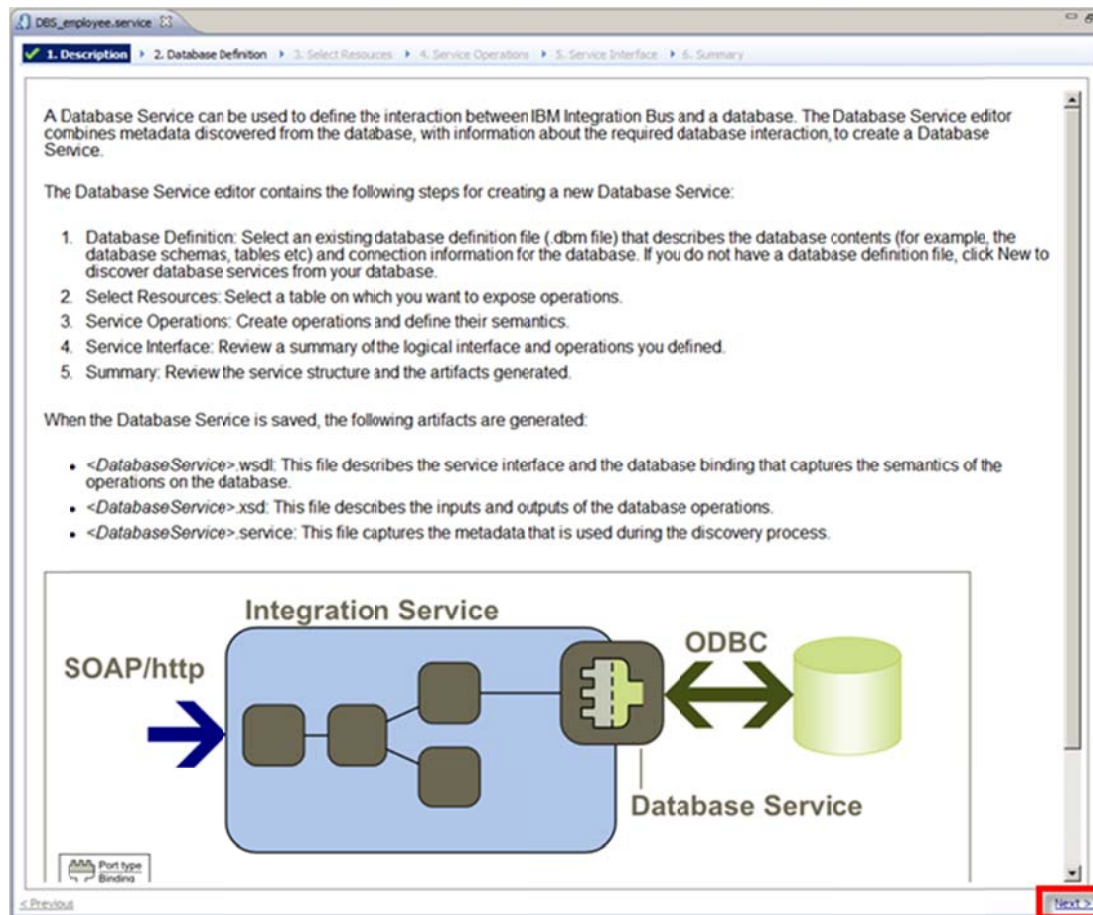
- ___1. Right-click on the **serviceDefinitions** Library and choose **New→Database Service**.



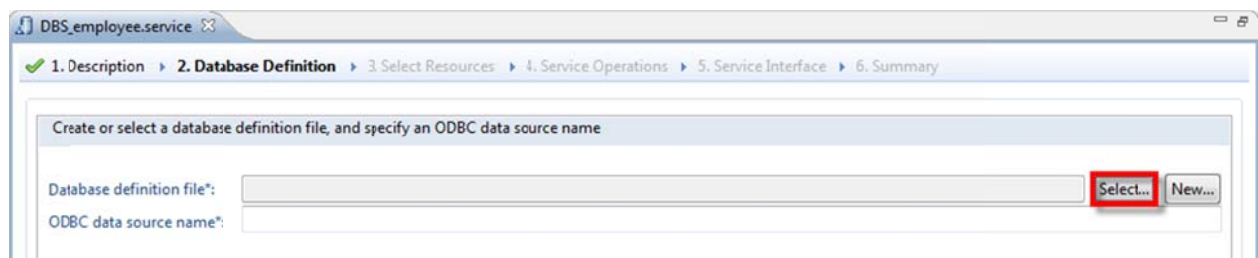
- ___2. Name the new service **DBS_employee**. Click **Finish**.



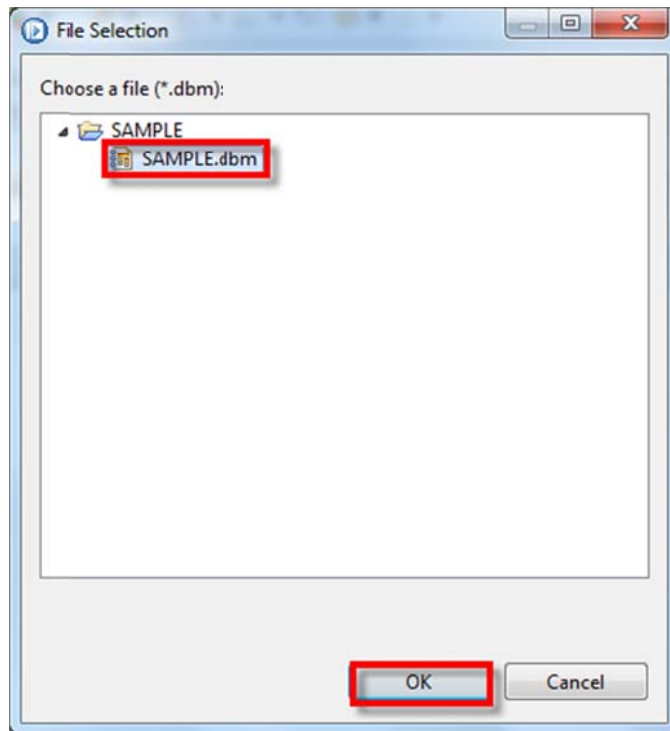
__3. The Database Service editor will open. Read the description and click **Next**.



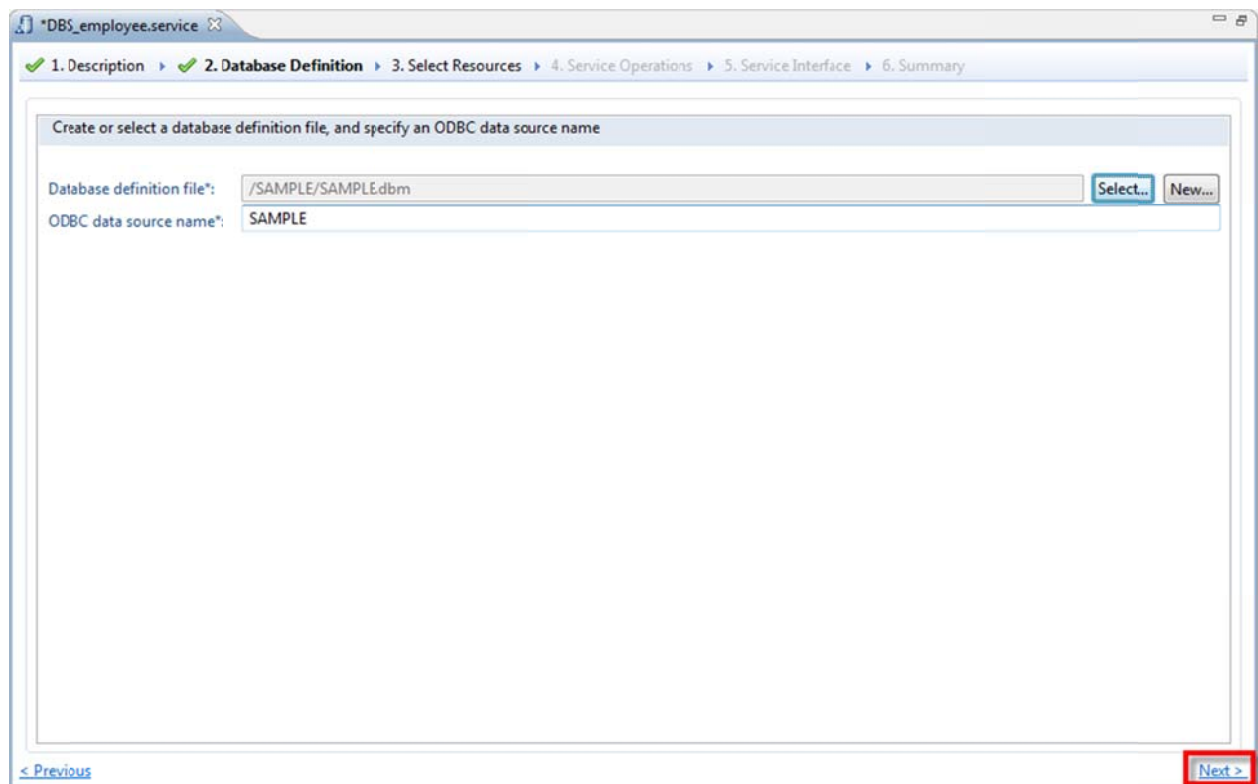
__4. On the Database Definition section, click the **Select...** button.



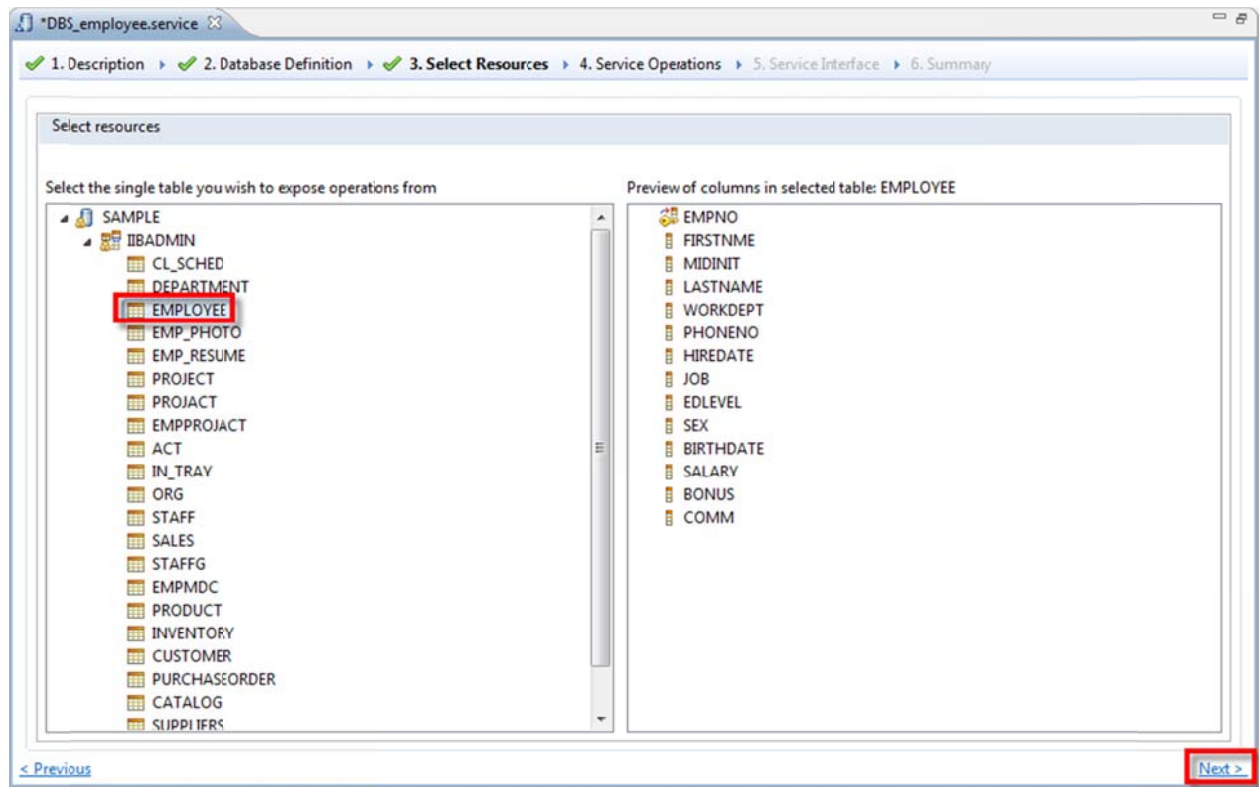
- ___5. Select the **SAMPLE.dbm** file and click **OK**.



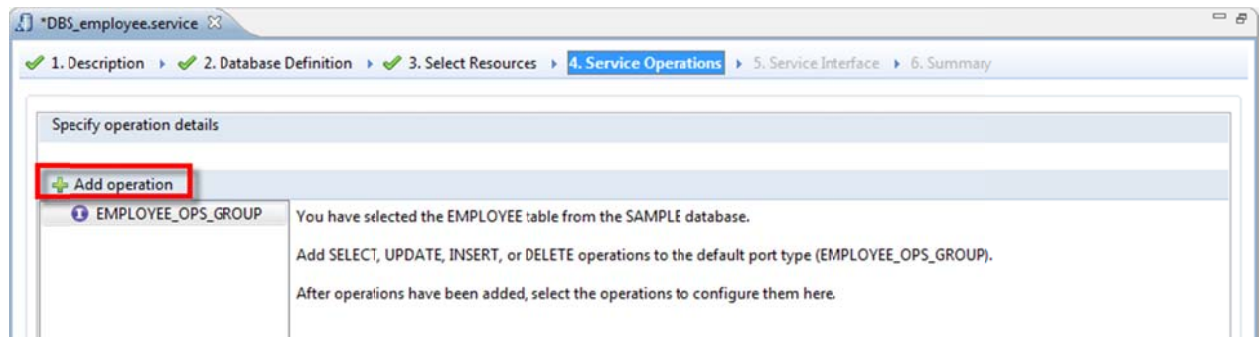
- ___6. The details of the SAMPLE.dbm file will be copied into the fields. Click **Next**.



- ___7. On the Select Resources page, click the **EMPLOYEE** table. Notice the preview of the table changes, giving an indication of the columns defined in the table. Click **Next**.



- ___8. On the Service Operations window, click **Add operation**.



- ___9. On the Add a database operation window, choose **SELECT** as the operation type and keep the default service operation name of **retrieveEmployee**. Click **OK**.

Add a database operation

Select operation type
SELECT

Service operation name
retrieveEmployee

On the selected table: EMPLOYEE

OK Cancel

- ___10. A list of column names that will be output parameters of running the **retrieveEmployee** operation will appear. **EMPNO** (defined as a key) will be selected by default. Right-click on the **EMPLOYEE** table and choose **Select All Columns**.

DBS_employee.service

1. Description 2. Database Definition 3. Select Resources 4. Service Operations 5. Service Interface 6. Summary

Specify operation details

Operation Type: SELECT -- Selected columns from the table will be output parameters of the operation. In the Output Columns tab set the sort preferences for the columns. In the Conditions tab set which rows will be returned.

Add operation

EMPLOYEE_OPS_GROUP
retrieveEmployee

EMPLOYEE

EMPNO
FIRSTNAME
MIDINIT
LASTNAME
WORKDEPT
PHONENO
HIREDATE
JOB
FTEFLVL

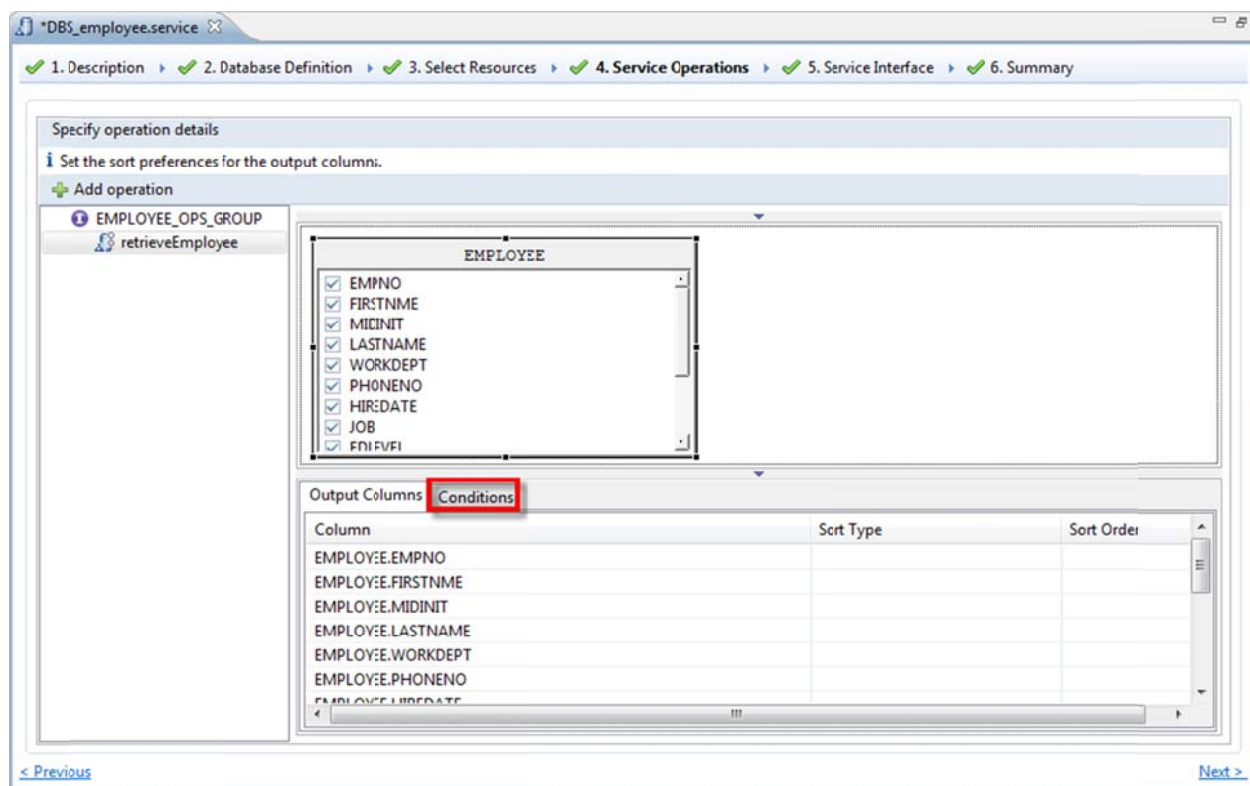
Select All Columns
Deselect All Columns

Output Columns Conditions

Column	Sort Type	Sort Order
EMPLOYEE.EMPNO		

< Previous Next >

__11. Click the **Conditions** tab.



- ___12. Click a blank entry under **Column**. From the drop down, choose **EMPLOYEE.EMPNO**. The **Operator** “=” and **Value** “:empno” (the input parameter) will be set automatically. Click **Next**.

DBS_employee.service

1. Description 2. Database Definition 3. Select Resources 4. Service Operations 5. Service Interface 6. Summary

Specify operation details

Operation Type: SELECT -- Selected columns from the table will be output parameters of the operation. In the Output Columns tab set the sort preferences for the columns. In the Conditions tab set which rows will be returned.

Add operation

EMPLOYEE_OPS_GROUP

retrieveEmployee

EMPLOYEE

- ☒ EMPNO
- ☒ FIRSTNAME
- ☒ MIDINIT
- ☒ LASTNAME
- ☒ WORKDEPT
- ☒ PHONENO
- ☒ HIREDATE
- ☒ JOB
- ☒ FNI :VFI

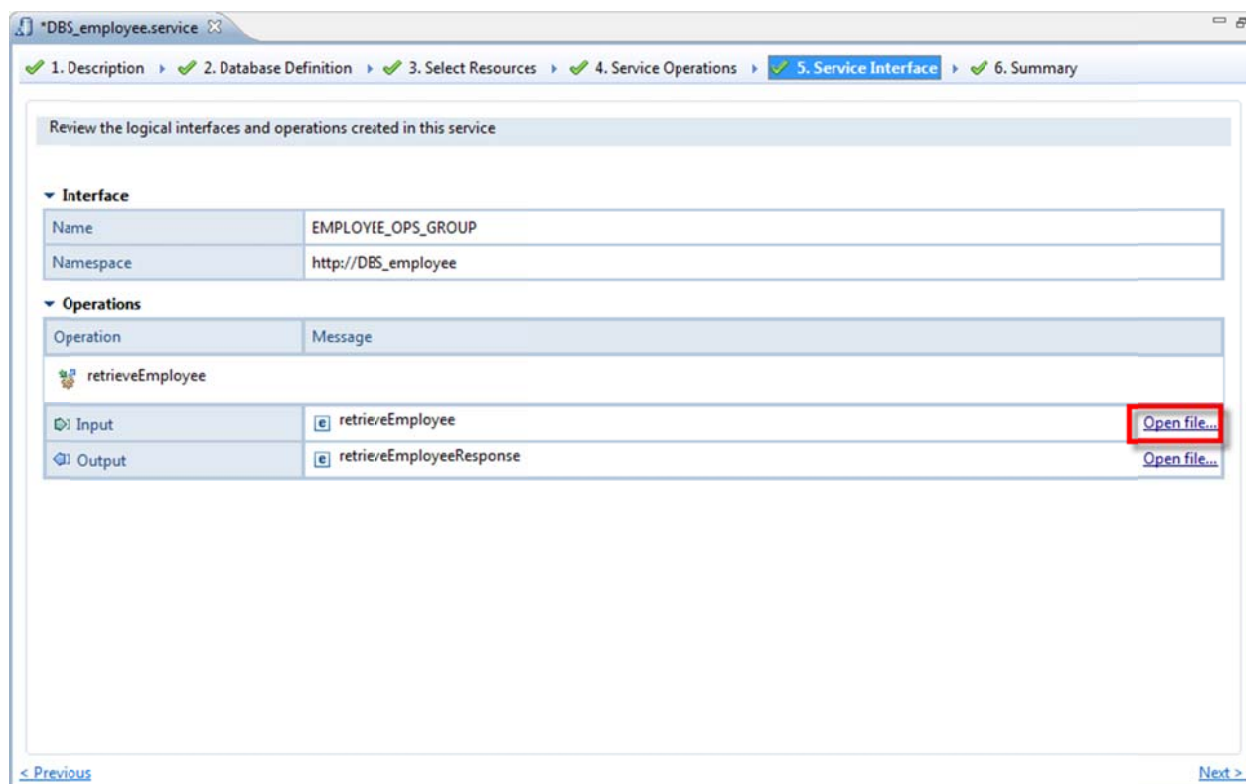
Output Columns Conditions

Column	Operator	Value	AND/OR
EMPNO	=	:empno	
EMPLOYEE.EMPNO			
EMPLOYEE.FIRSTNAME			
EMPLOYEE.MIDINIT			
EMPLOYEE.LASTNAME			
EMPLOYEE.WORKDEPT			

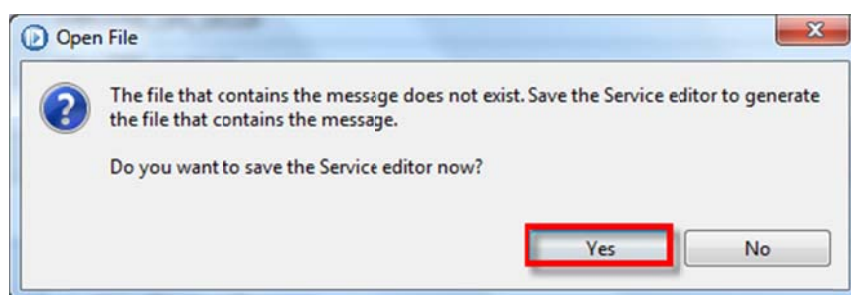
< Previous

Next >

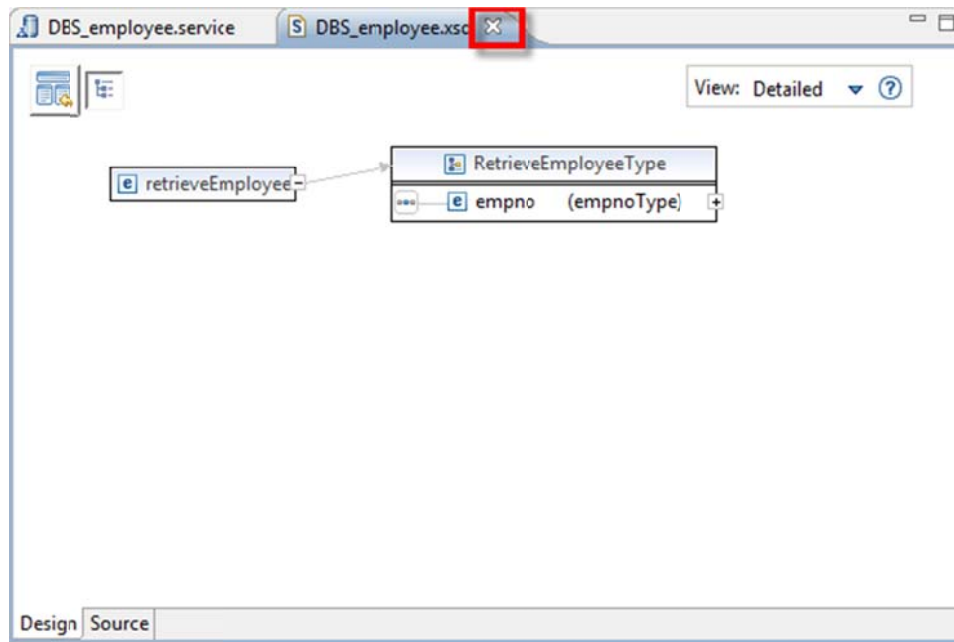
- ___13. On the 5. Service Interface screen, click the **Open file...** link on the Input - **retrieveEmployee** column.



- ___14. A message will appear indicating that the (xsd) file does not exist. Click **Yes** to save the editor file and generate the file.



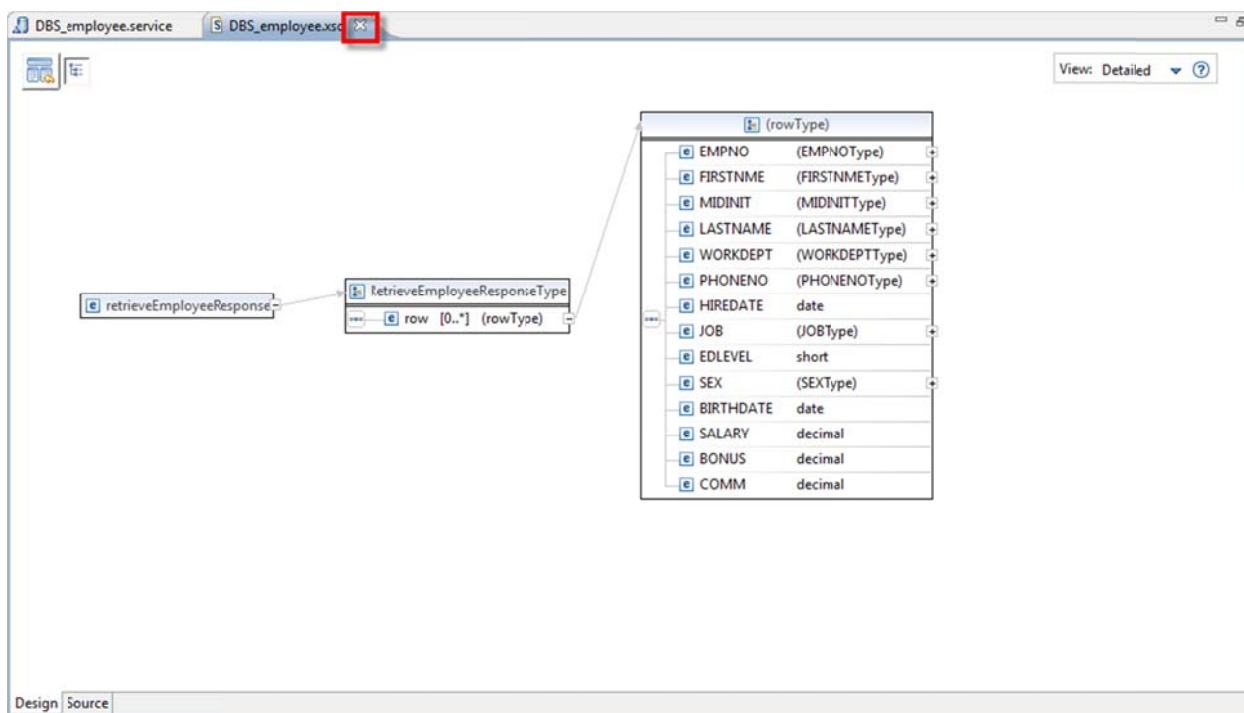
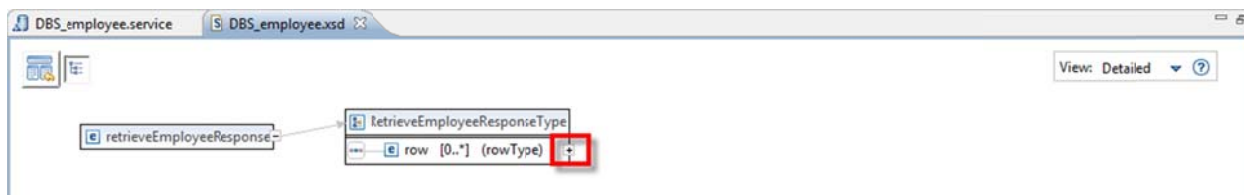
- ___15. Once generated, the retrieveEmployee (input) xsd file (**DBS_employee.xsd**) will be shown. Close the **DBS_employee.xsd** file (click on the **X**).



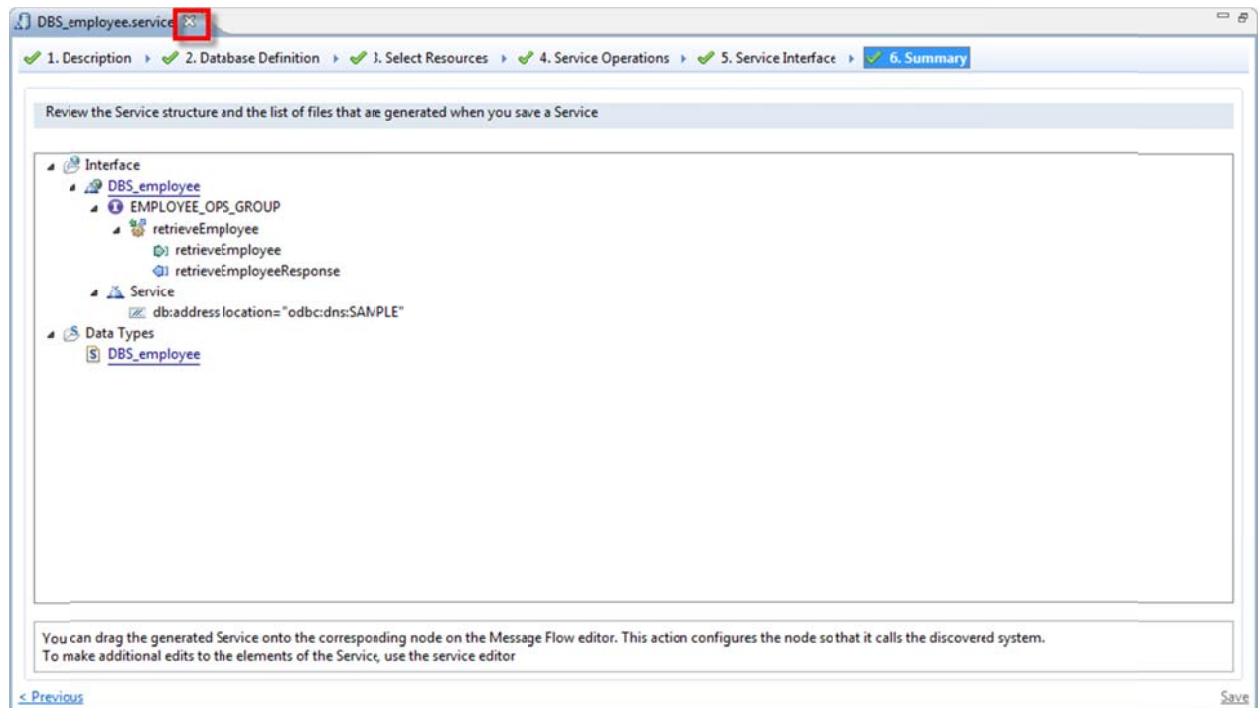
- ___16. Clicking on the **retrieveEmployeeResponse Open file...** link will also show the generated xsd file for the output (note the columns you choose to be output are included). Click the **+** sign in the schema to get the list. Close the **DBS_employee.xsd** file.

Operations

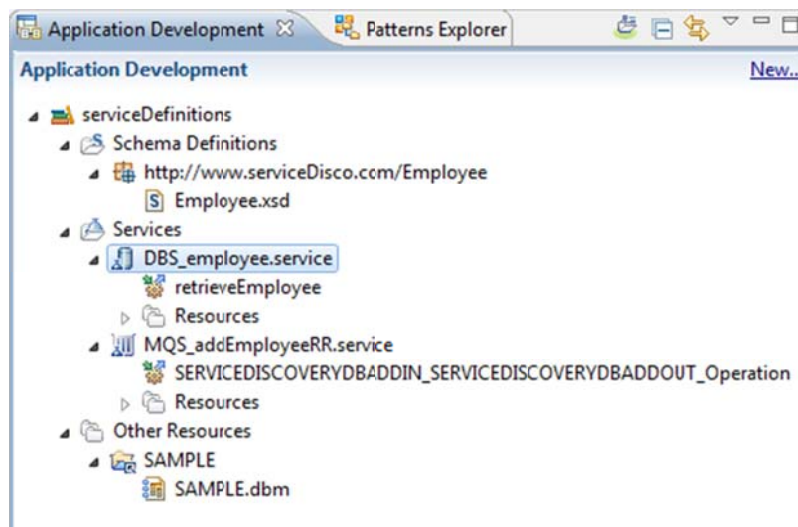
Operation	Message
retrieveEmployee	
Input	retrieveEmployee
Output	retrieveEmployeeResponse



- ___17. Click **Next** to see the 6. Summary page. Since you saved the file to generate the DBS_employee.xsd file, the save button will be inactive since there have been no changes since the last save. Close the **DBS_employee.service** editor.



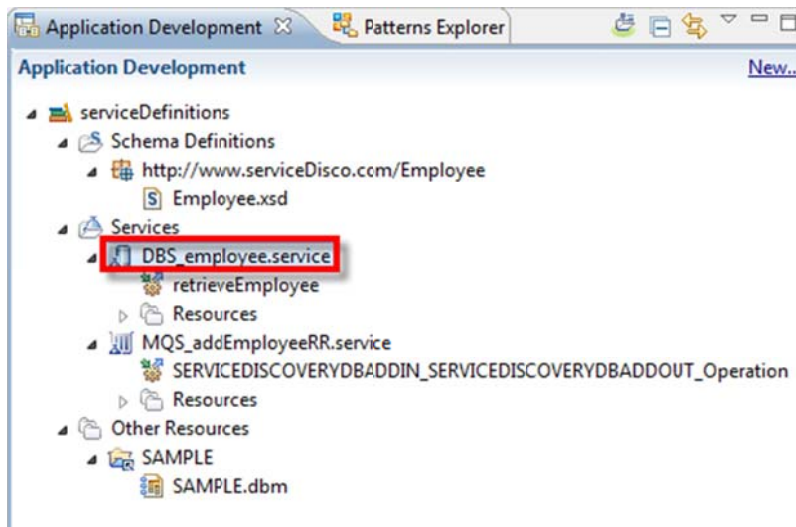
- ___18. Note now that you have created the Database Service, the Navigator has been updated to include the Database Service in the list of services in the serviceDefinitions Library.



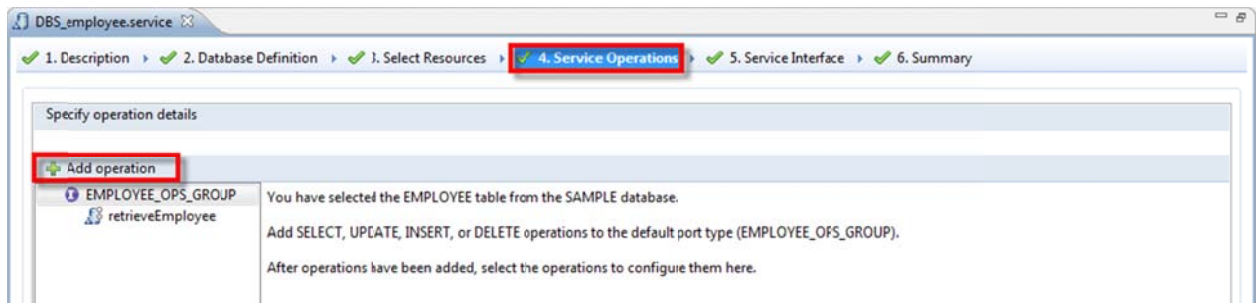
5.4.2 Add an operation to the Database Service

Additional database operations can be added to a discovered Database Service. The application that you will create later in this lab will add and retrieve data in the database. When you discovered the Database Service you added the “Select” operation. This will be used to retrieve data from the database by the application. You will now edit the existing discovered Database Service and add the Insert operation so that you can add data to the database using the Database Service.

- ___1. Open the Database Service editor by double clicking on **DBS_employee.service** in the serviceDefinitions Library.



- ___2. In the Database Service editor, click **4. Service Operations**. Note the retrieveEmployee operation that we created earlier. Click **+ Add Operation**.



- ___3. Choose **INSERT** from the Select operation type drop down. The service operation name will be set to **createEmployee**. Click **OK**.

Add a database operation

Select operation type
INSERT

Service operation name
createEmployee

On the selected table: EMPLOYEE

OK Cancel

- ___4. As before, right-click on the **EMPLOYEE** table and chose **Select All Columns**.

DBS_employee.service

1. Description 2. Database Definition 3. Select Resources 4. Service Operations 5. Service Interface 6. Summary

Specify operation details

Operation Type: INSERT -- Selected columns from the table will be input parameters of the operation. In the Specify Values tab, specify the values for the columns to insert.

Add operation

EMPLOYEE_OPS_GROUP

- retrieveEmployee
- createEmployee

EMPLOYEE

EMPNO
FIRSTNAME
MIDINIT
LASTNAME
WORKDEPT
PHONENO
HIREDATE
JOB
FBI FIVEI

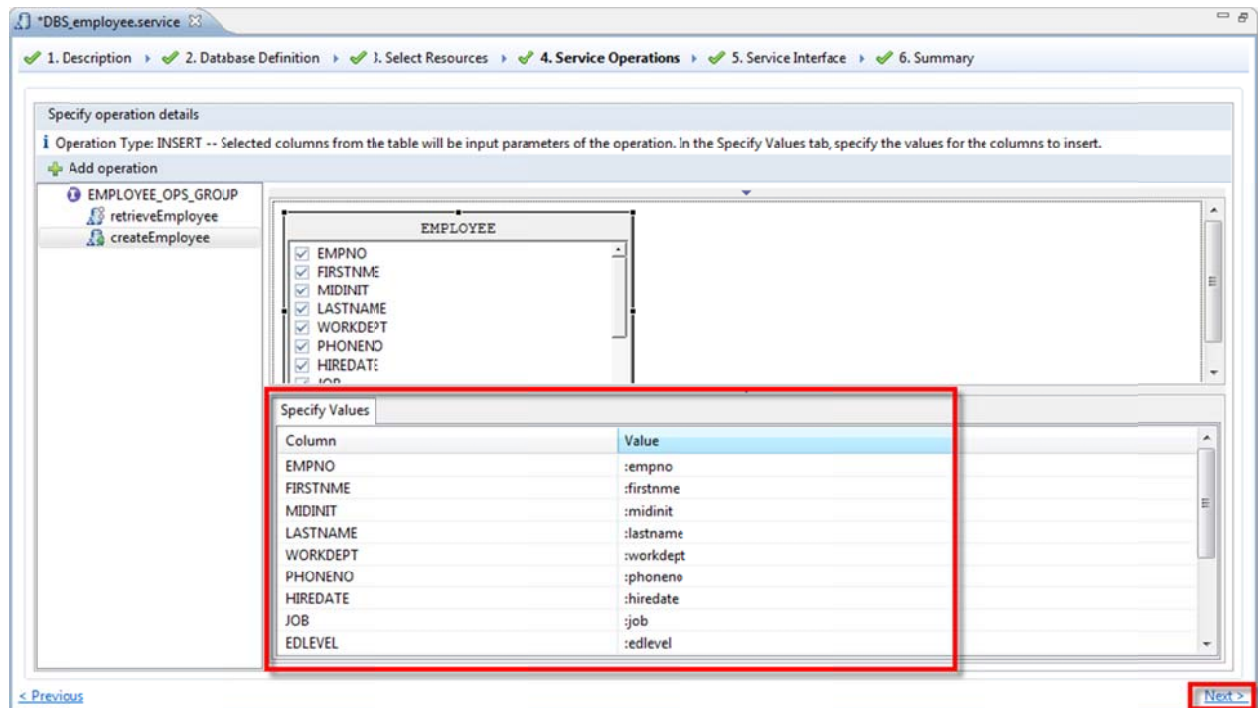
Select All Columns
Deselect All Columns

Specify Values

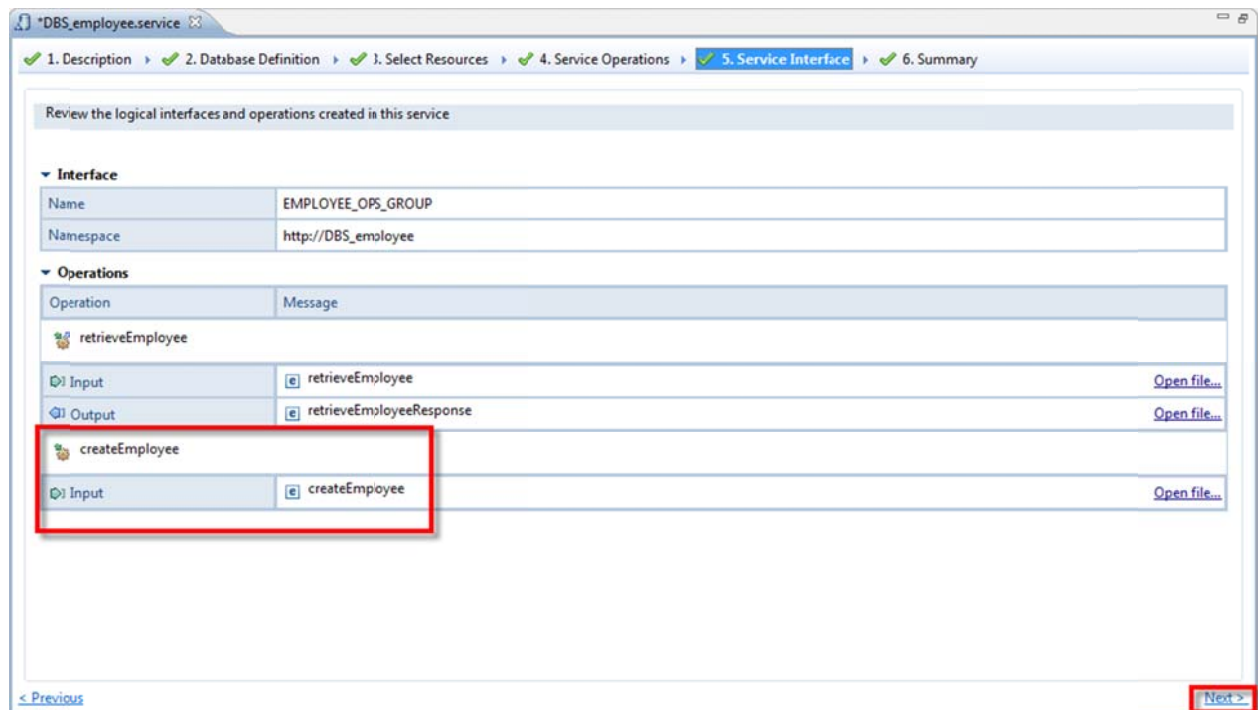
Column	Value
EMPNO	:empno

< Previous Next >

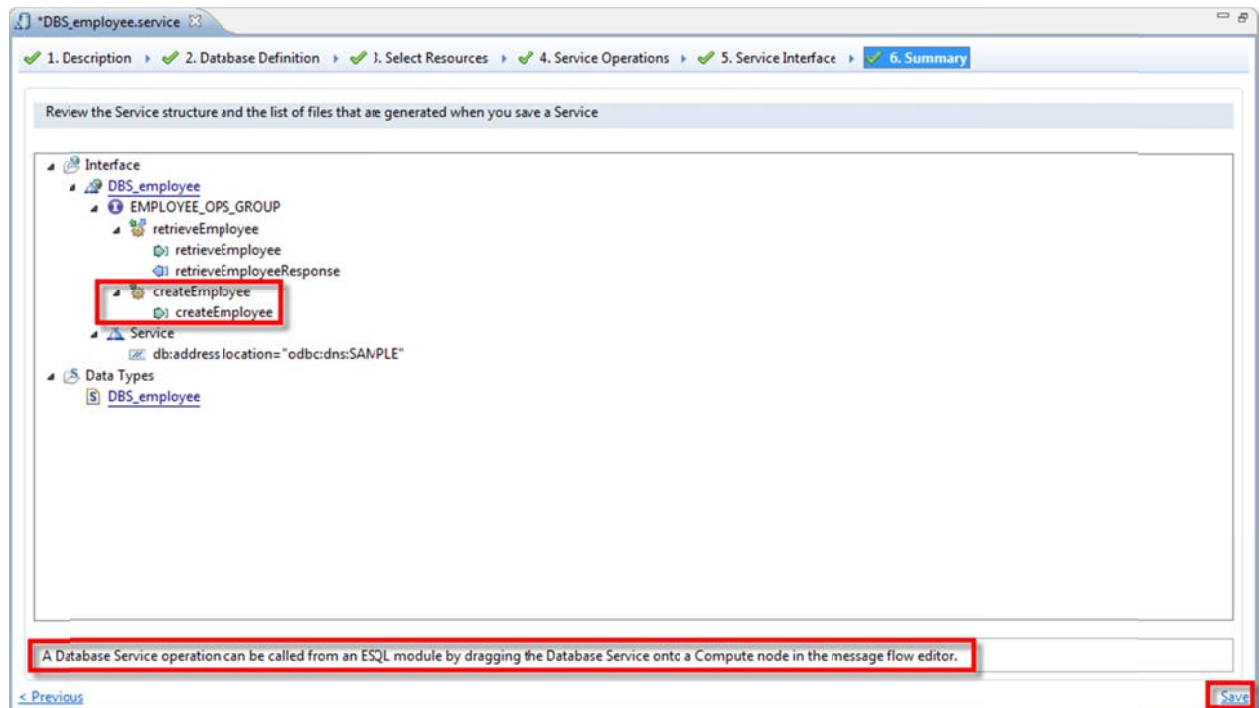
__5. The values in the columns fields will be automatically generated. Click **Next**.



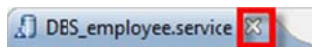
__6. The **createEmployee** operation will be added to the Service Interface section. Click **Next**.



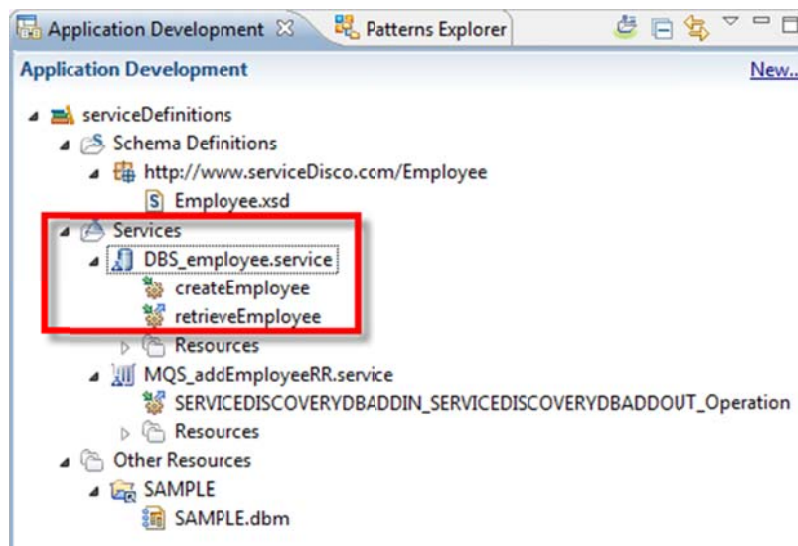
- ___7. On the summary screen, note the **createEmployee** Operation has been added to the EMPLOYEE_OPS_GROUP and the explanation on how to use the Database Service. Click **Save** to add the createEmployee operation to the Database Service.



- ___8. Close the service editor.



- ___9. You will now see the **createEmployee** operation in the Navigator.

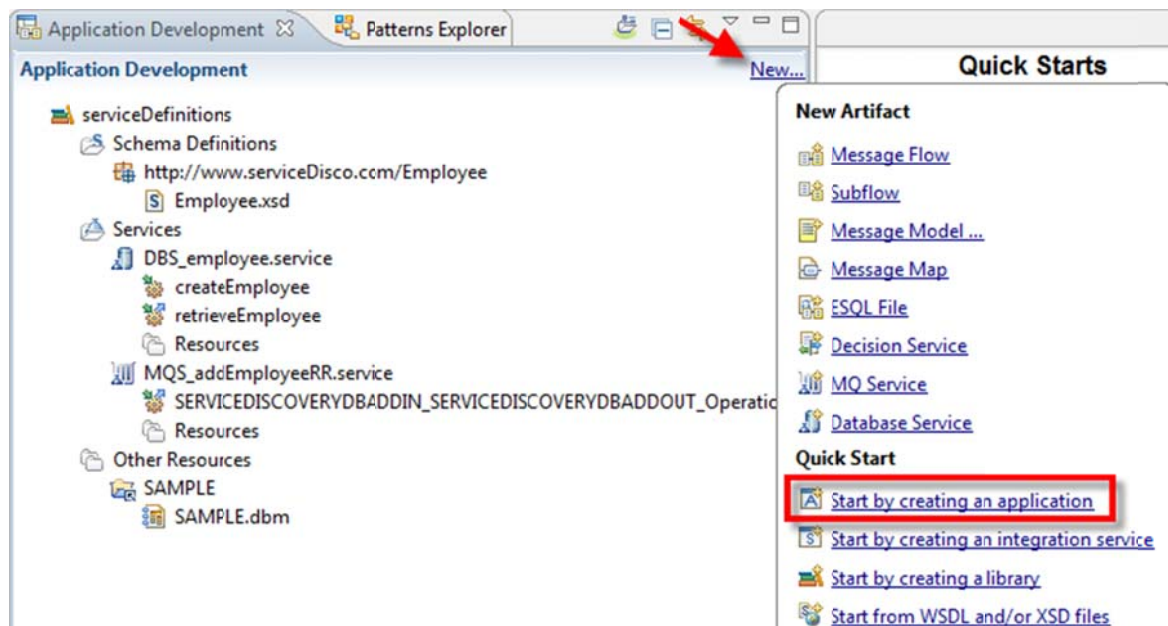


5.5 Using discovered services

You will now create an application that will use both the MQ request/response service and Database Service that you just created. MQ Services can be called or implemented when using them in an Integration Bus message flow. The application you will create in the next few sections will demonstrate how each of these options can be used to create an application based on the MQ Service and Database Service.

5.5.1 Create an application

1. Create a new application. Click **New...** and then select **Start by creating an application** from the menu.



- __2. Name the application **ManageEmployee**. Click **Next**.

The screenshot shows the 'New Application' dialog box with the title 'Create a new application'. Below the title, it says 'An application is a deployable container that provides isolation at runtime. Enter a name for the new application.' The 'Application name' text box contains 'ManageEmployee', which is highlighted with a red rectangle. At the bottom, the 'Next >' button is also highlighted with a red rectangle. Other buttons include '< Back', 'Finish', and 'Cancel'.

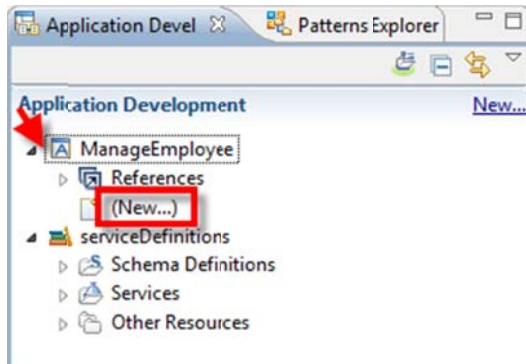
- __3. Create a reference from this application to the library that contains the services you defined, by checking the box for **serviceDefinitions**. Click **Finish**.

The screenshot shows the 'New Application' dialog box with the title 'Create a new application'. Below the title, it says 'Specify dependencies on libraries.' The section 'Select existing libraries to reference from the new Application:' contains a list with 'serviceDefinitions' checked, highlighted by a red rectangle. Below this, it says 'The following libraries will also be included as they are referenced by the checked libraries.' At the bottom, the 'Finish' button is highlighted with a red rectangle. Other buttons include '< Back', 'Next >', and 'Cancel'.

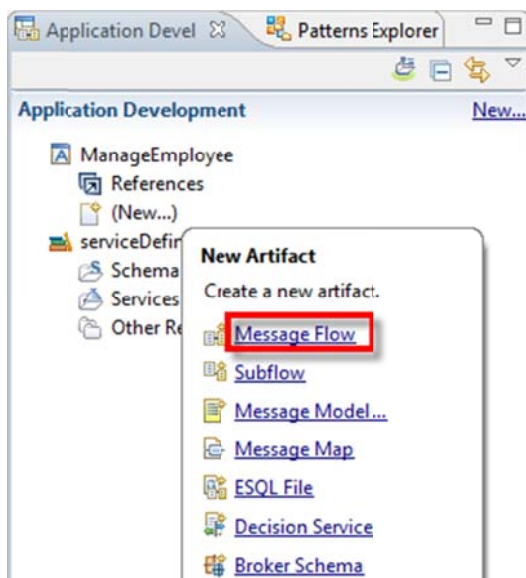
5.5.2 Create a flow to implement the MQ Service

The flow you will now create will use the MQ_addEmployeeRR service you created in order to **implement** the MQ Service. You will also use the Database Service you created earlier in this flow.

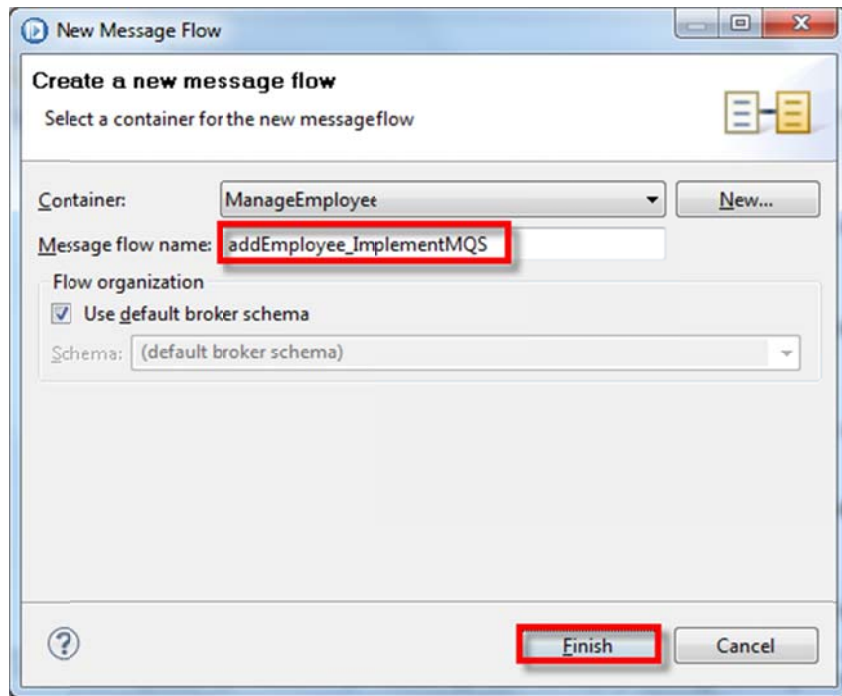
- __1. In the Navigator, expand the **ManageEmployee** application and click **(New ...)**,



- __2. Select **Message Flow** from the list of options.

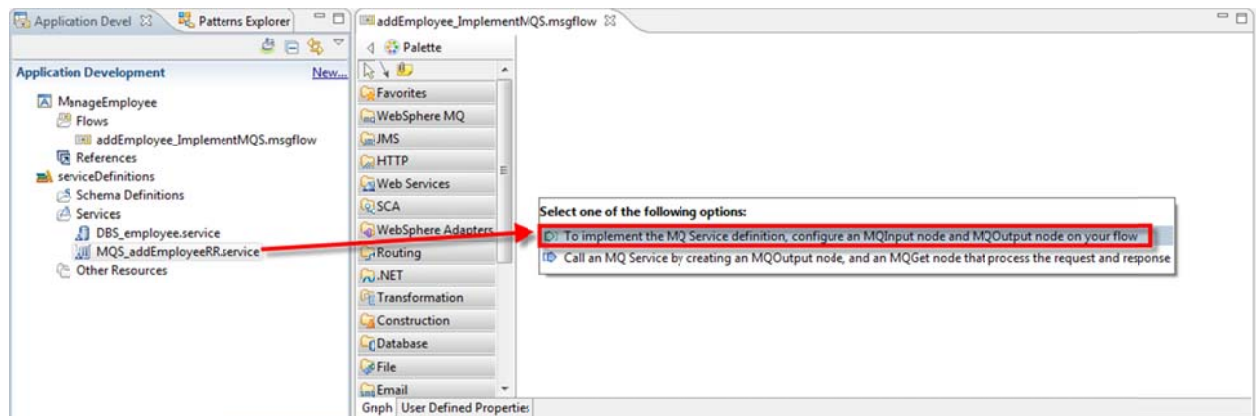


- __3. Name the message flow **addEmployee_ImplementMQS** and click **Finish**.

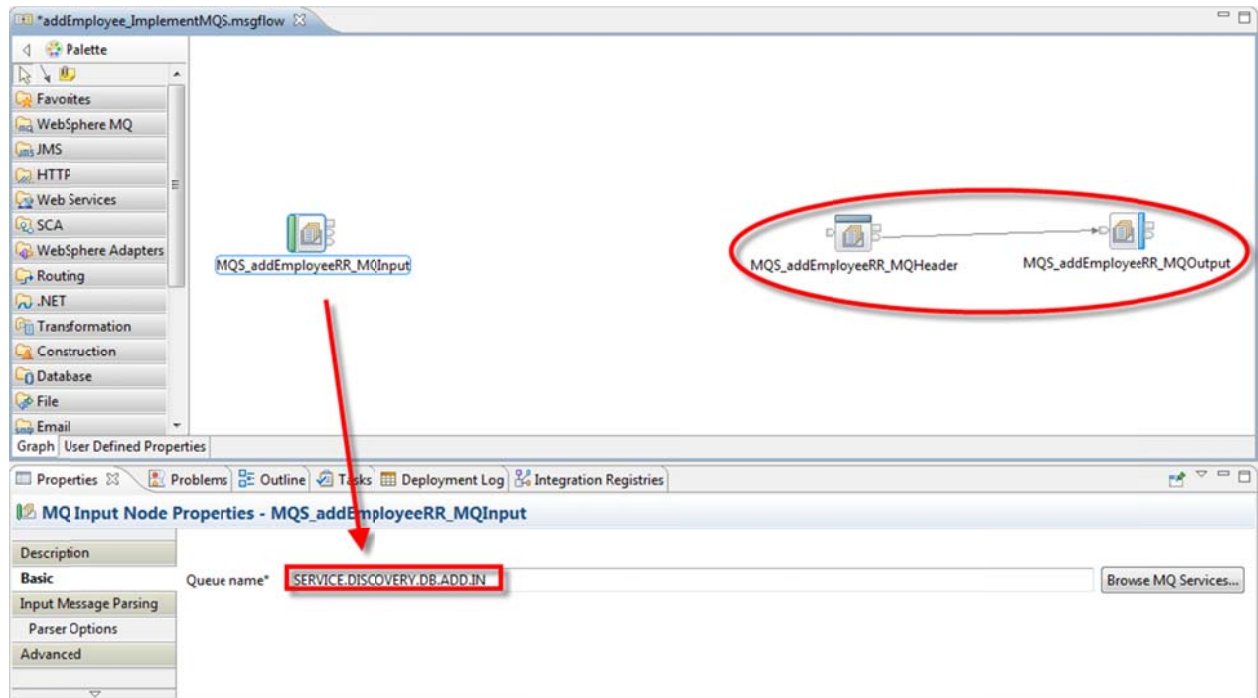


- __4. The message flow editor will open. In the **serviceDefinitions** Library, drag and drop **MQS_addEmployeeRR.service** from the Navigator onto the editor canvas.

When prompted, choose **To implement the MQ Service definition, configure an MQInput node and MQOutput node on your flow**.



- ___5. Note that when implementing the service, the following are automatically created:
- ___a. an MQ Input node (specified with the queue name **SERVICE.DISCOVERY.DB.ADD.IN**)
 - ___b. an MQ Header wired to an MQ Output node (specified with **SERVICE.DISCOVERY.DB.ADD.OUT**)

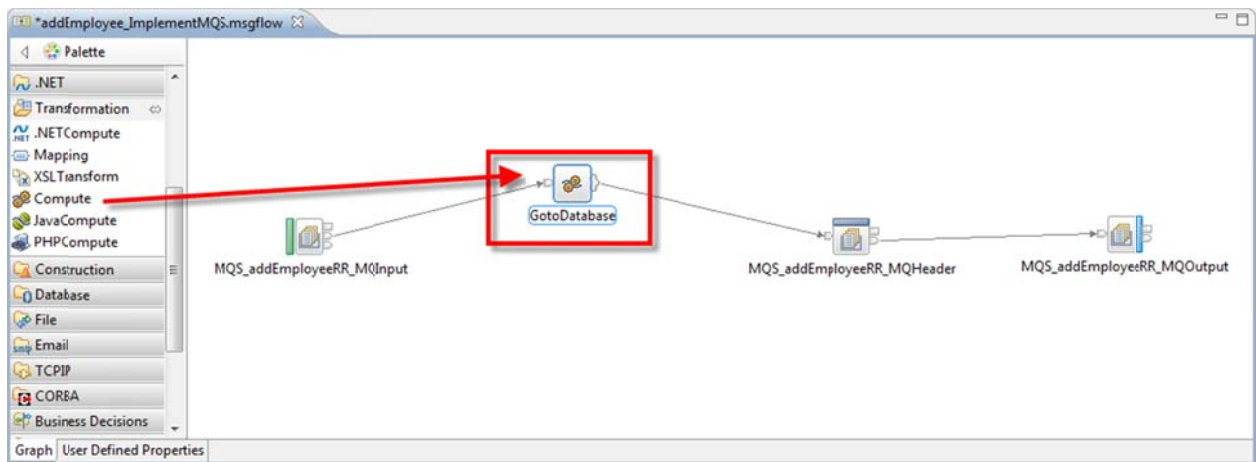


Since we are now implementing the MQ Service, the message flow has been designed for request / response. This message flow accepts the request on the input queue and responds to the caller on the output queue. You will now use the Database Service to create the request.

- ___6. Drag and drop a **Compute** node (in the Transformation drawer) onto the canvas. Name the node **GotoDatabase**.

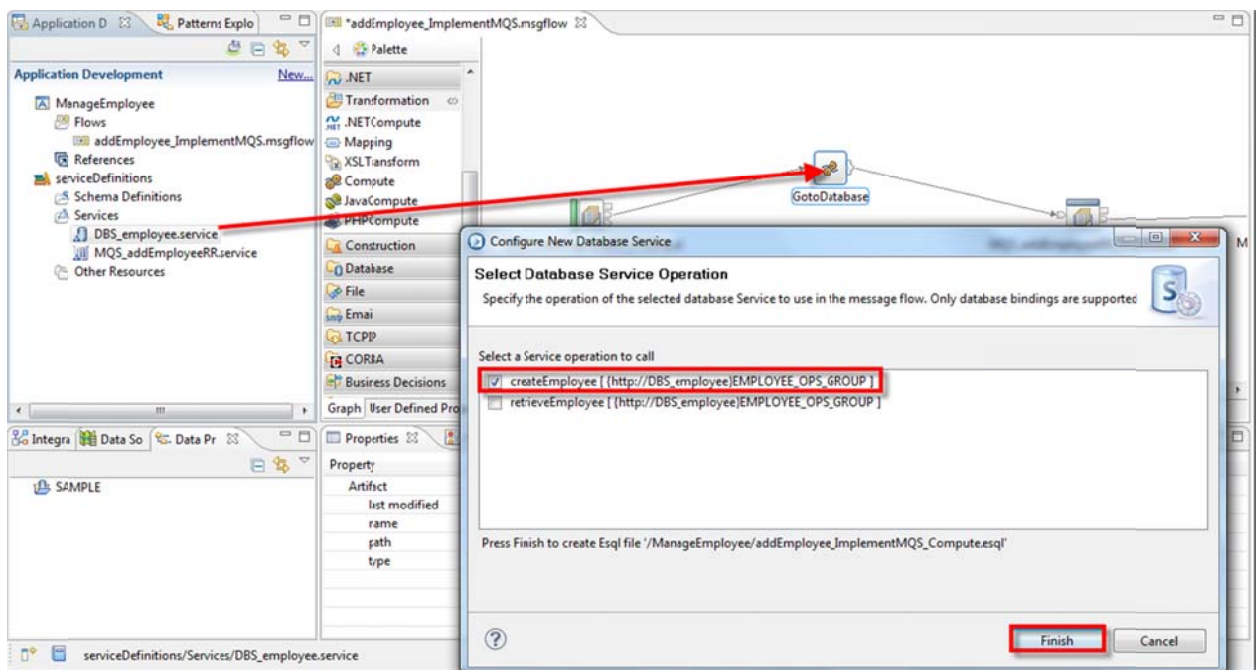
Wire the **Out** terminal on the MQ input node (**MQS_addEmployeeRR_MQInput**) to the **In** terminal on the **GotoDatabase** node.

Wire the **Out** terminal of the **GotoDatabase** node to the **In** terminal of the MQ header node (**MQS_addEmployeeRR_MQHeader**).



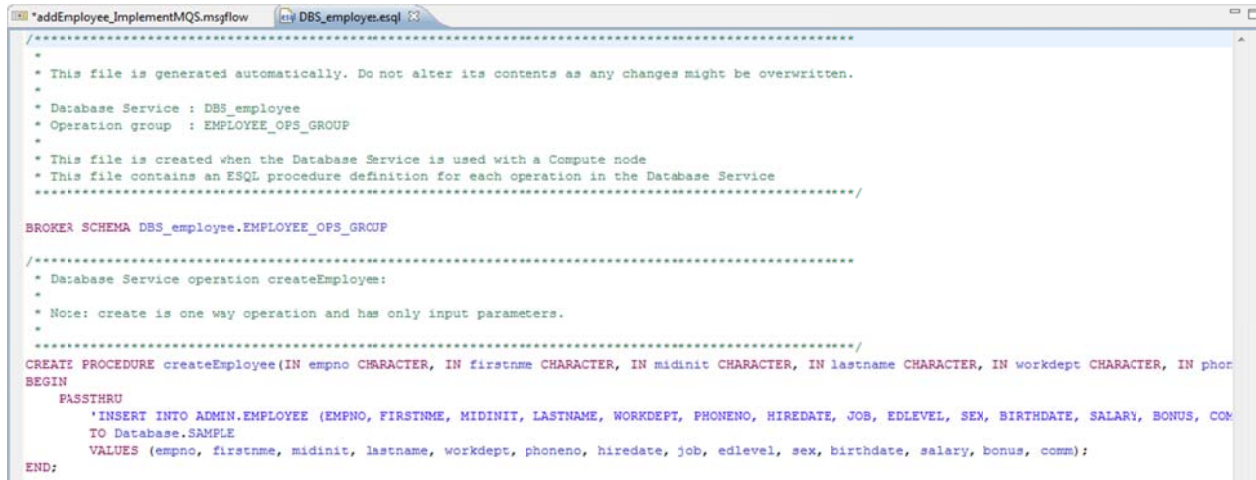
- ___7. Drag and drop the Database Service **DBS_employee** onto the compute node **GotoDatabase**.

When prompted to select the Database Service Operation, two options will be given: createEmployee and retrieveEmployee (you created these operations when you created the Database Service earlier). Select **createEmployee** and click **Finish**.



- __8. Dragging and dropping the Database Service definition on the Compute (ESQL) node automatically creates an ESQL procedure named **createEmployee**.

The ESQL procedure is automatically generated and saved in the library. It is saved in a file called **DBS_employee.esql**. The toolkit automatically opens this file.



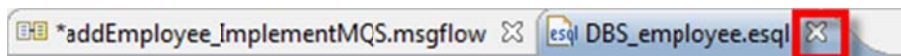
```
*****
* This file is generated automatically. Do not alter its contents as any changes might be overwritten.
*
* Database Service : DBS_employee
* Operation group : EMPLOYEE_OPS_GROUP
*
* This file is created when the Database Service is used with a Compute node
* This file contains an ESQL procedure definition for each operation in the Database Service
*****/

BROKER SCHEMA DBS_employee.EMPLOYEE_OPS_GROUP

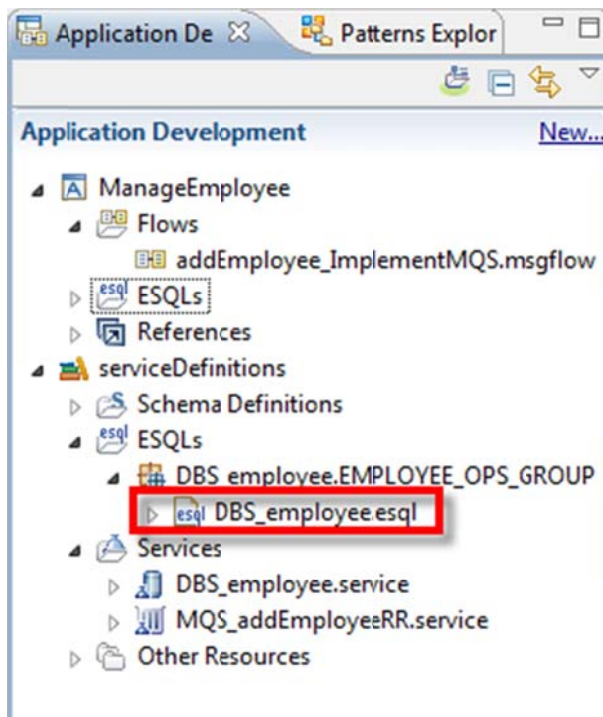
/*****
* Database Service operation createEmployee:
*
* Note: create is one way operation and has only input parameters.
*
*****/

CREATE PROCEDURE createEmployee(IN empno CHARACTER, IN firstnme CHARACTER, IN midinit CHARACTER, IN lastname CHARACTER, IN workdept CHARACTER, IN phoneno CHARACTER, IN hiredate DATE, IN job CHARACTER, IN edlevel CHARACTER, IN sex CHARACTER, IN birthdate DATE, IN salary DECIMAL(10,2), IN bonus DECIMAL(10,2), IN comm DECIMAL(10,2))
BEGIN
  PASSTHRU
  'INSERT INTO ADMIN.EMPLOYEE (EMPNO, FIRSTNME, MIDINIT, LASTNAME, WORKDEPT, PHONENO, HIREDATE, JOB, EDLEVEL, SEX, BIRTHDATE, SALARY, BONUS, COMM)
  TO Database.SAMPLE
  VALUES (empno, firstnme, midinit, lastname, workdept, phoneno, hiredate, job, edlevel, sex, birthdate, salary, bonus, comm);
END;
```

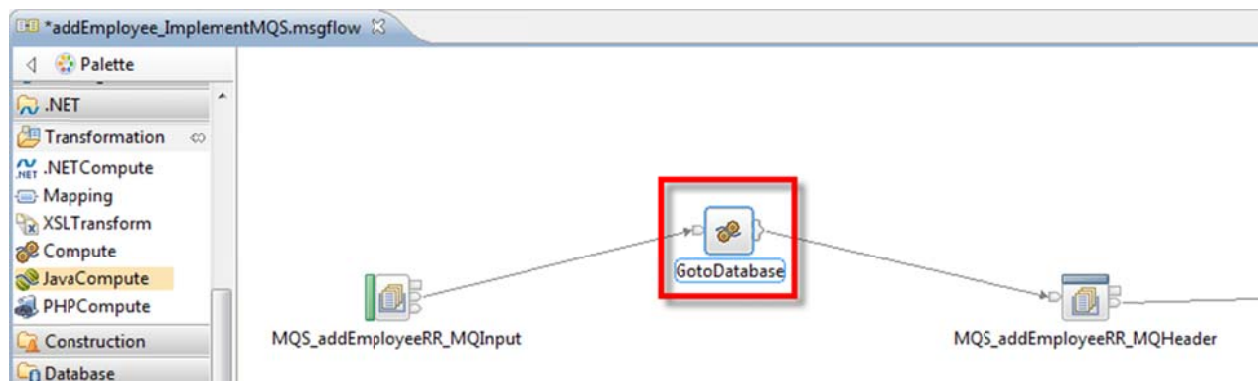
- __9. Close this file (DBS_employee.esql).



If you need to see the procedure definition again for any reason, the file is saved in the **serviceDefinitions** Library.



- __10. Go back to the message flow again and double click on the **GotoDatabase** Compute node.



- __11. This will open some sample “starter” ESQL code that can be used to drive the Compute node.

```
addEmployee_ImplementMQS.msgflow  esql addEmployee_ImplementMQS_Compute.esql
PATH DBS_employee.EMPLOYEE_OPS_GROUP;

CREATE COMPUTE MODULE addEmployee_ImplementMQS_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- CALL CopyMessageHeaders();
    -- CALL CopyEntireMessage();
    RETURN TRUE;
END;

CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER;
    SET J = CARDINALITY(InputRoot.*[]);
    WHILE I < J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I + 1;
    END WHILE;
END;

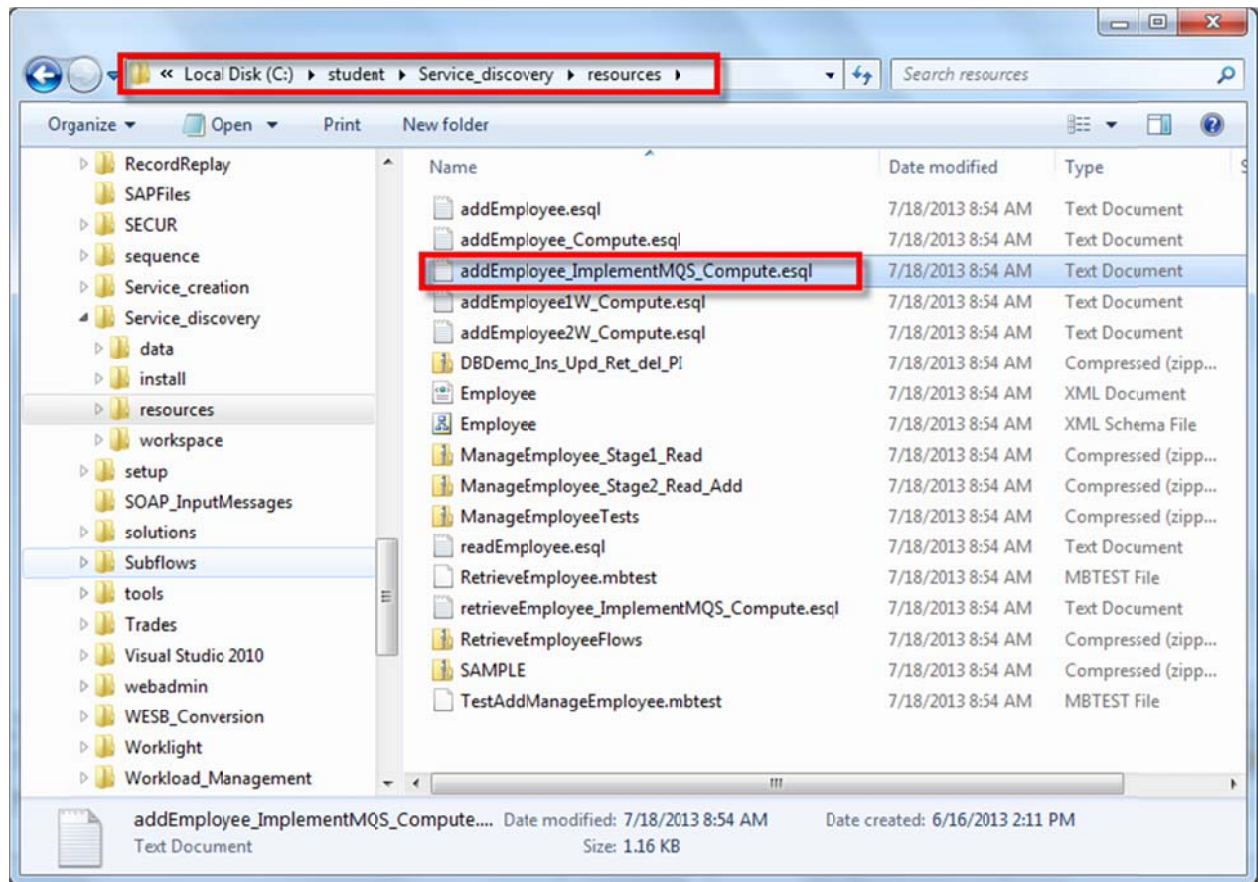
CREATE PROCEDURE CopyEntireMessage() BEGIN
    SET OutputRoot = InputRoot;
END;

END MODULE;
```

We will not be using this sample “starter” ESQL. We have supplied some code that will call the ESQL procedure that was automatically created by dragging and dropping the Database Service on the compute node.

- __12. Highlight all of the sample code (**CTRL-A**) and **delete** all the lines of code (in the next section you will copy some new code into it). Keep this file open in the Toolkit ready to be filled with code.

- ___13. In **Windows Explorer**, navigate to **C:\student\Service_discovery\resources** and open the file **addEmployee_ImplementMQS_Compute.esql.txt** with **Notepad** (double-click on the file).



___14. Select (**CTRL-A**) and copy (**CTRL-C**) all the code from this file.

```
addEmployee_ImplementMQS_Compute.esql - Notepad
File Edit Format View Help
PATH DBS_employee.EMPLOYEE_OPS_GROUP;
DECLARE ns NAMESPACE 'http://www.serviceDisco.com/Employee';

CREATE COMPUTE MODULE addEmployee_ImplementMQS_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- CALL CopyMessageHeaders();
    CALL CopyEntireMessage();

    DECLARE insRef REFERENCE TO InputRoot.XMLNSC.ns:Employee;

    DECLARE EMP1 CHAR insRef.EmpNo;

    CALL createEmployee(EMP1,
        insRef.FirstName,
        insRef.Initial,
        insRef.LastName,
        insRef.Dept,
        insRef.Phone,
        CAST('2013-05-13' AS DATE),
        insRef.Job, CAST (insRef.EdLevel AS INTEGER),
        insRef.Sex,
        CAST('1959-11-17' AS DATE),
        CAST(insRef.Salary AS DECIMAL),
        CAST(insRef.Bonus AS DECIMAL),
        CAST(insRef.Commission AS DECIMAL));

    RETURN TRUE;
END;

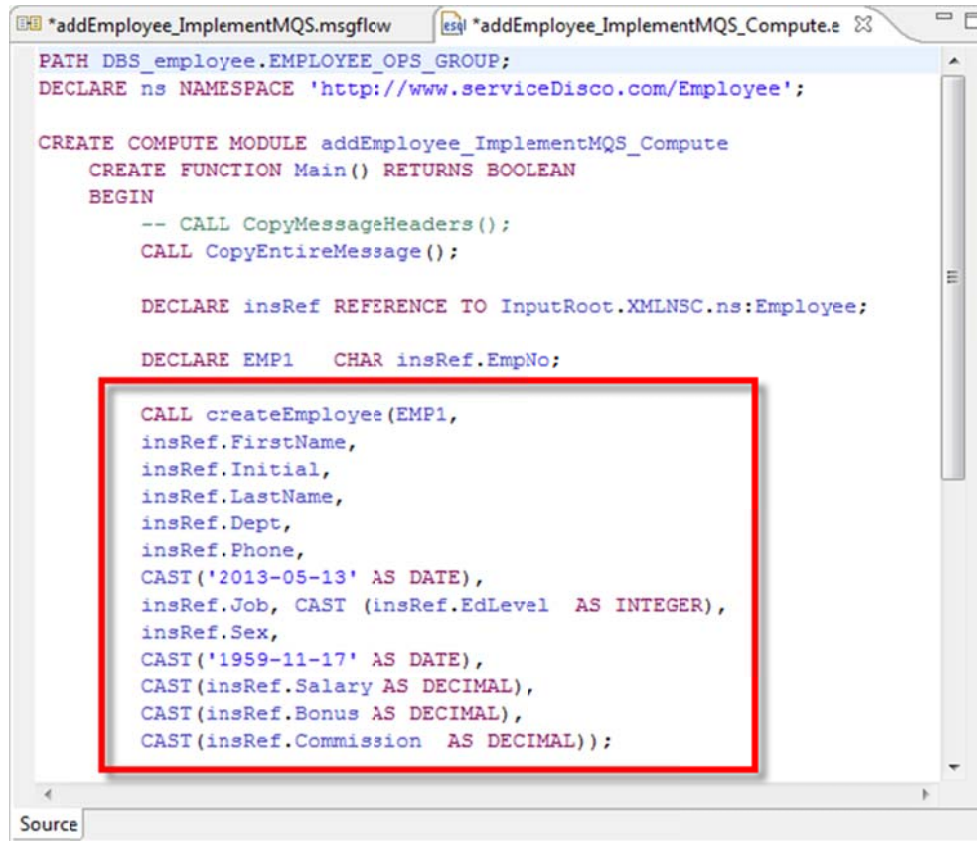
CREATE PROCEDURE CopyMessageHeaders() BEGIN
    DECLARE I INTEGER 1;
    DECLARE J INTEGER;
    SET J = CARDINALITY(InputRoot.*[I]);
    WHILE I < J DO
        SET OutputRoot.*[I] = InputRoot.*[I];
        SET I = I + 1;
    END WHILE;
END;

CREATE PROCEDURE CopyEntireMessage() BEGIN
    SET OutputRoot = InputRoot;
END;

END MODULE;
```

- ___15. Go back to the Toolkit and paste (**press CTRL-V**) the lines of code into **addEmployee_ImplementMQS_Compute.esql** (you left this file open ready to be filled with code).

The file will look like this: (note the call to **createEmployee** in the code).



```
PATH DBS_employee.EMPLOYEE_OPS_GROUP;
DECLARE ns NAMESPACE 'http://www.serviceDisco.com/Employee';

CREATE COMPUTE MODULE addEmployee_ImplementMQS_Compute
CREATE FUNCTION Main() RETURNS BOOLEAN
BEGIN
    -- CALL CopyMessageHeaders();
    CALL CopyEntireMessage();

    DECLARE insRef REFERENCE TO InputRoot.XMLNSC.ns:Employee;

    DECLARE EMP1 CHAR insRef.EmpNo;

    CALL createEmployee(EMP1,
        insRef.FirstName,
        insRef.Initial,
        insRef.LastName,
        insRef.Dept,
        insRef.Phone,
        CAST('2013-05-13' AS DATE),
        insRef.Job, CAST (insRef.EdLevel AS INTEGER),
        insRef.Sex,
        CAST('1959-11-17' AS DATE),
        CAST(insRef.Salary AS DECIMAL),
        CAST(insRef.Bonus AS DECIMAL),
        CAST(insRef.Commission AS DECIMAL));
```

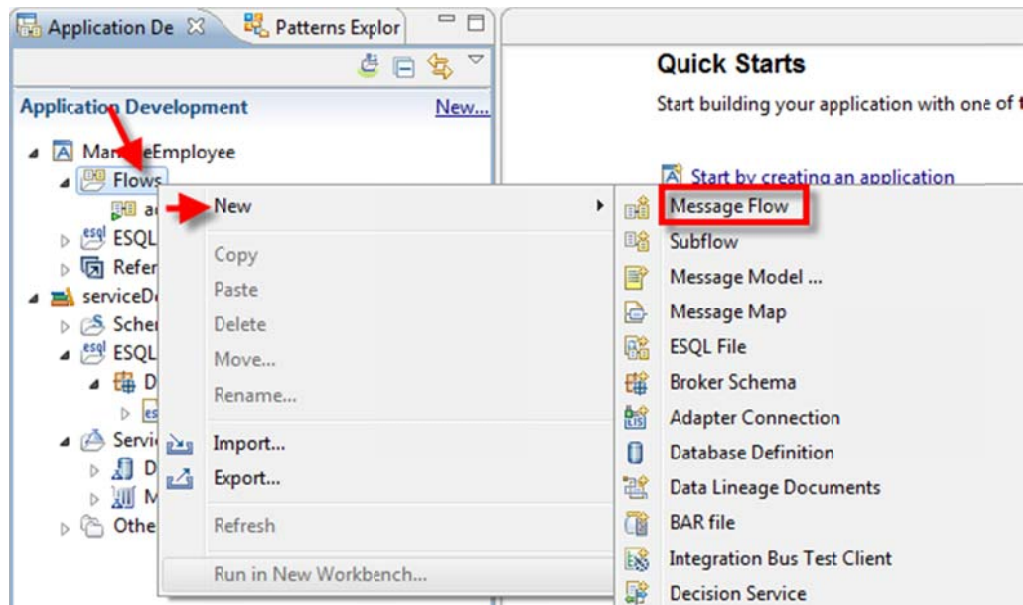
You can close Notepad.

- ___16. Save (**CTRL-S**) and close the **addEmployee_ImplementMQS_Compute.esql** file.
- ___17. Save (**CTRL-S**) the **addEmployee_ImplementMQS** message flow.
- ___18. Close the **addEmployee_ImplementMQS** message flow.

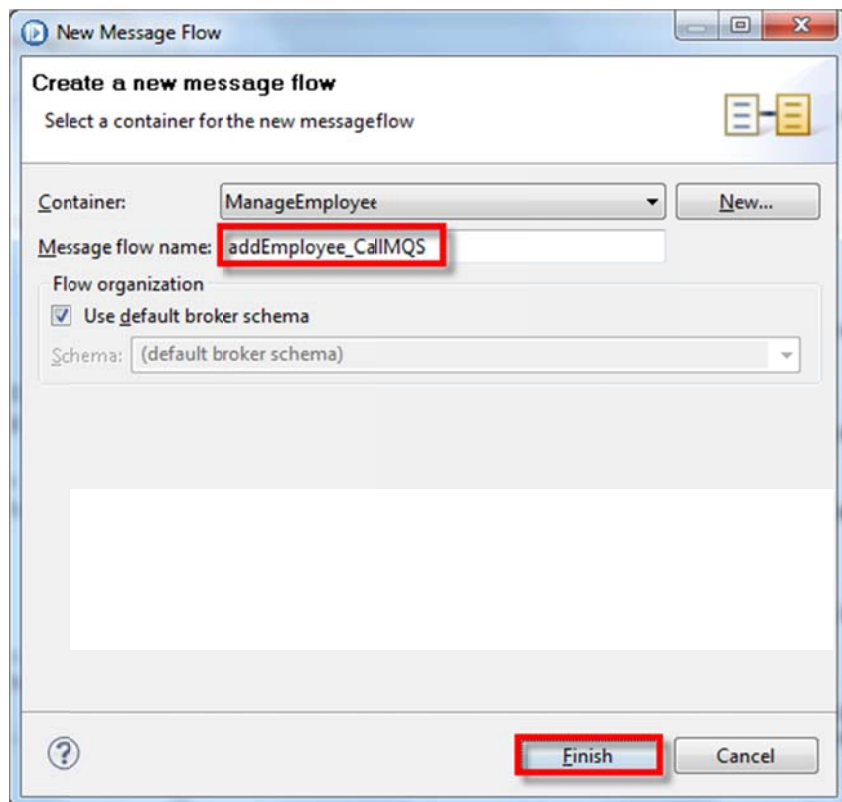
5.5.3 Create a flow to call the MQ Service

The flow you will now create will use the MQ_addEmployeeRR service you created in order to **call** the MQ Service.

- ___1. In the Navigator, expand the **ManageEmployee** application. Highlight the **Flows** folder, right-click, and select **New**→**Message Flow** from the menu.

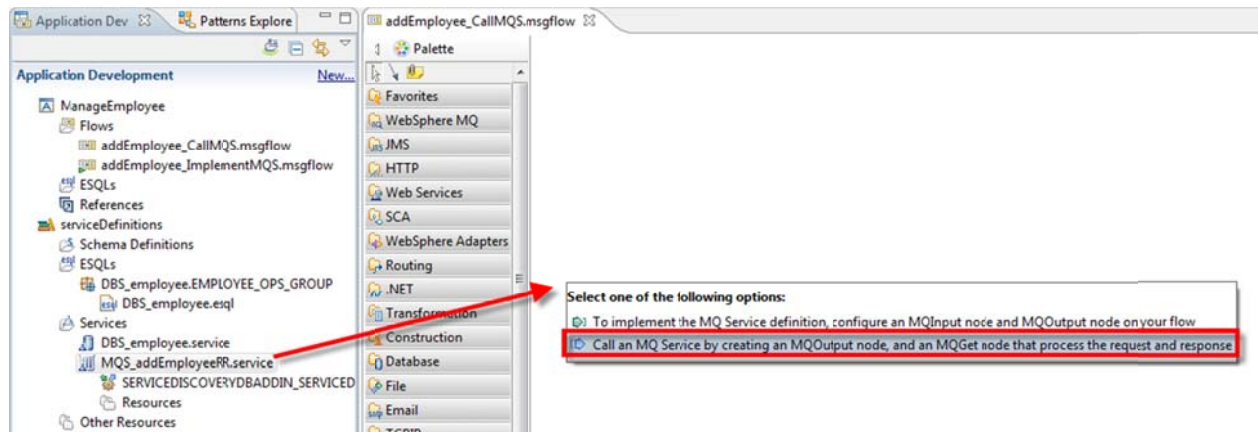


- ___2. Name the message flow **addEmployee_CallMQS** and click **Finish**.

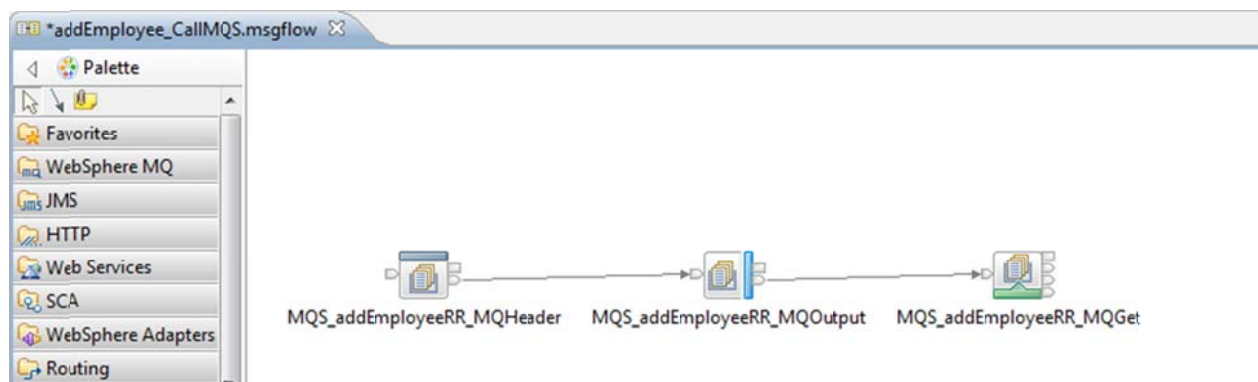


The message flow editor will open.

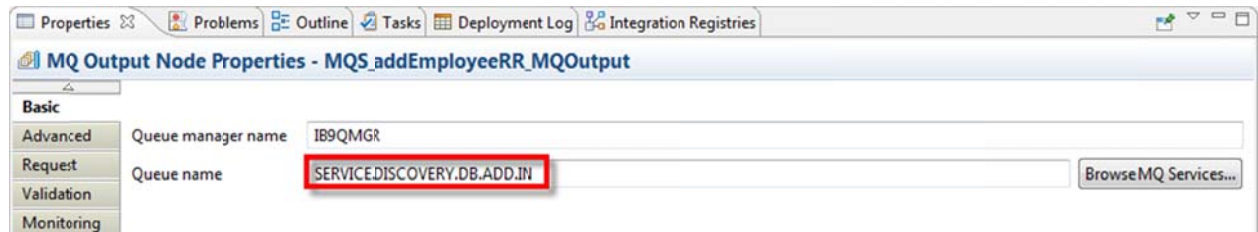
- ___3. From the **serviceDefinitions** Library in the Navigator, drag and drop the MQ Service **MQS_addEmployeeRR.service** onto the message flow canvas. When dropped onto the canvas you will be presented with two options – click the option to **Call an MQ Service by creating an MQOutput node, and an MQGet node that process the request and response**.



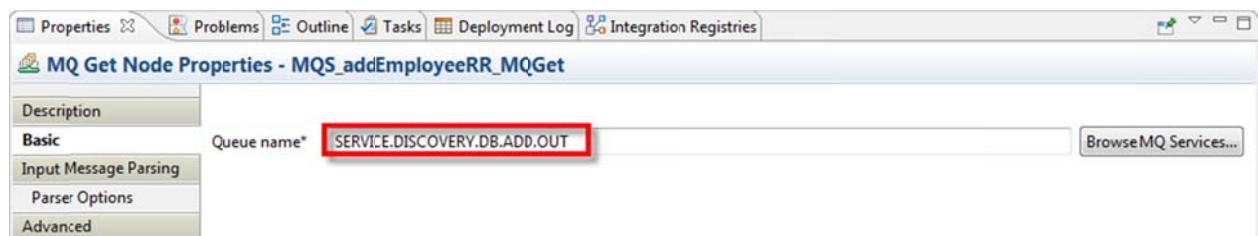
- ___4. Three MQ nodes will be created on the canvas.



- ___5. Note the MQ Output node has been pre-filled with the **Request** queue you defined in the MQ service earlier.



The MQ Get node has been pre-populated with the queue you specified as the **Response** queue.

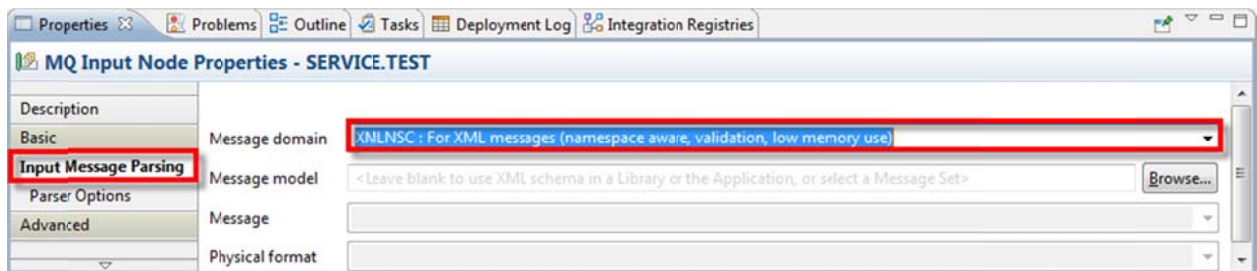
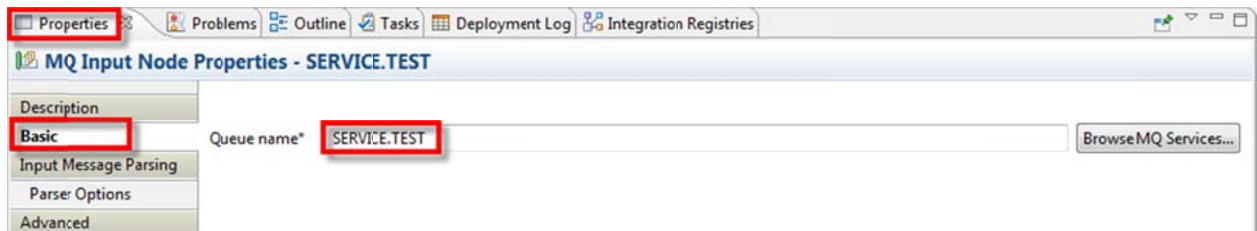
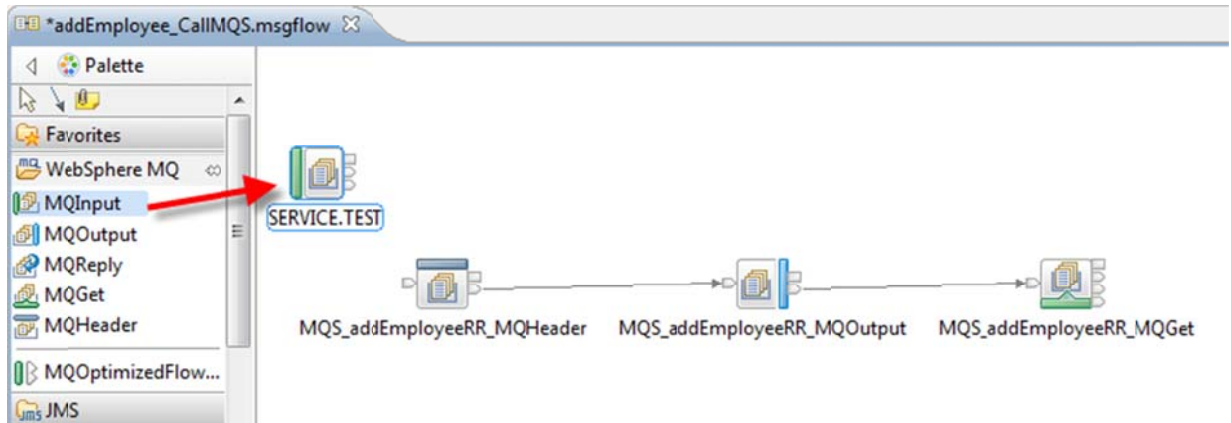


- ___6. The message flow will write to the SERVICE.DISCOVERY.DB.ADD.IN queue and wait until a response is received on the SERVICE.DISCOVERY.DB.ADD.OUT queue (thereby calling the MQ Service to add a user).

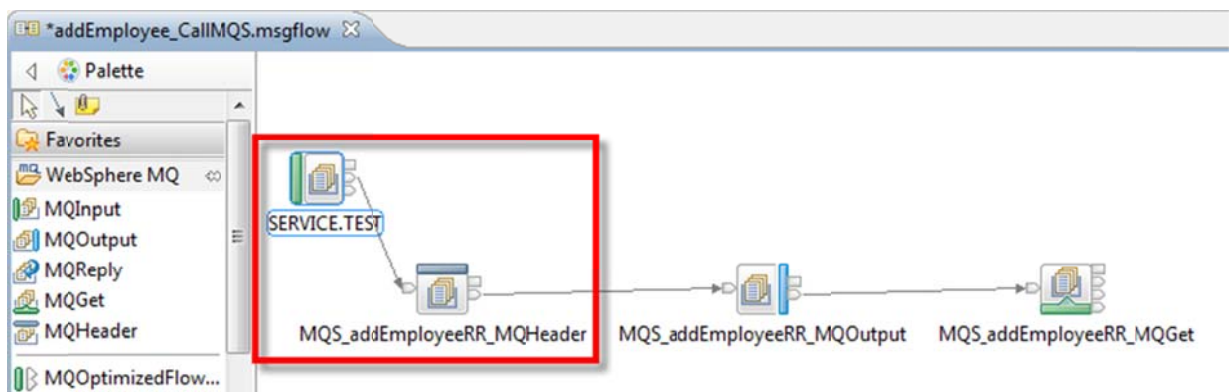
In order to test this message flow, you will now add an MQ Input node and a File Output node so that the message flow can run in your environment.

- ___7. Drag and drop an **MQInput** node (from the **WebSphere MQ** drawer) onto the canvas.

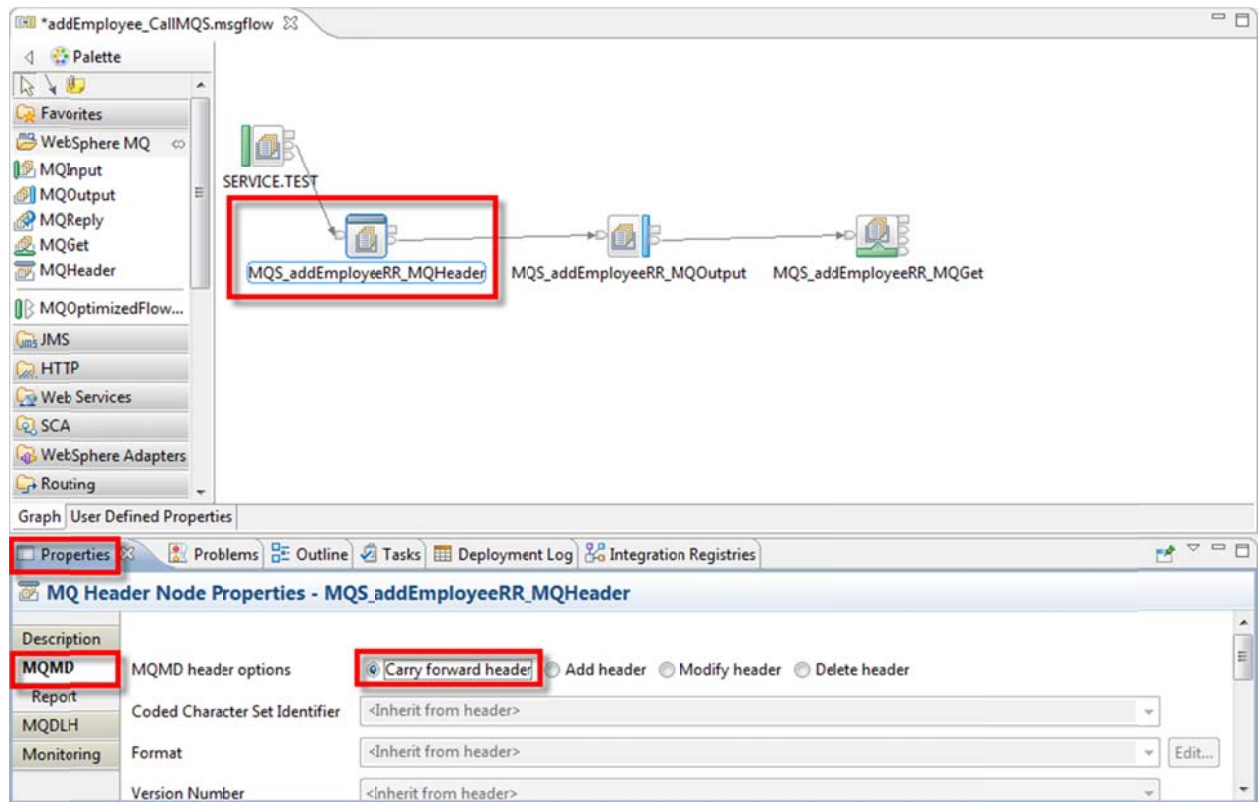
In the Properties, name the node **SERVICE.TEST**, set the input queue name to **SERVICE.TEST** (on the Basic tab) and the Input Message Parsing to **XMLNSC** (on the **Input Message Parsing** tab).



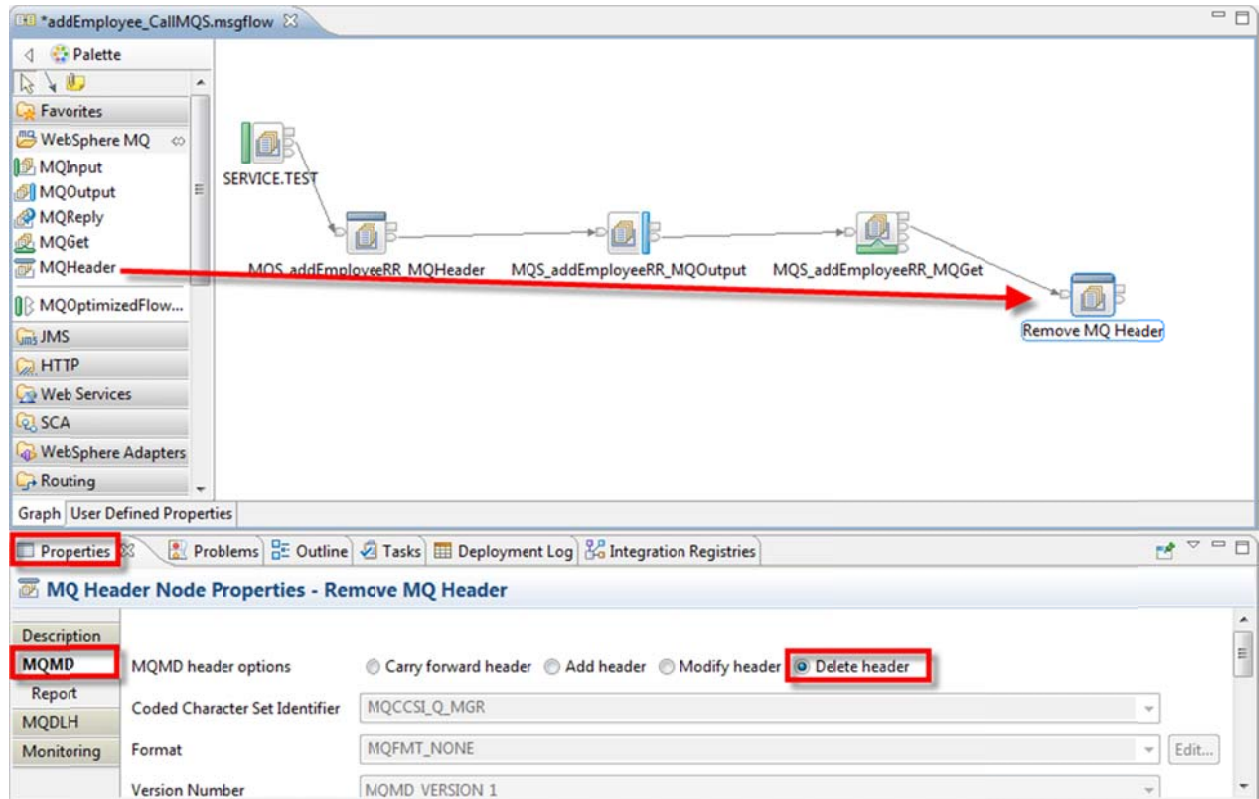
- ___8. Wire the **Out** terminal on the **SERVICE.TEST** MQ input node to the **In** terminal on the **MQS_addEmployeeRR_MQHeader** node.



- ___9. The default setting for the MQHeader node that is created by using the service is to add an MQ header. Change this setting to **Carry forward header** in the **Properties (MQMD tab)**.



- ___10. Drag an **MQHeader** node onto the Canvas, dropping the node after the MQGet node. Rename the node to **Remove MQ Header**. Wire the **Out** terminal of the **MQGet** node called **MQS_addEmployeeRR_MQGet** (added automatically by using the service definition) to the **In** terminal of the **Remove MQ Header** node. In the **Properties** of the Header node, set **Delete Header** (on the **MQMD** tab).



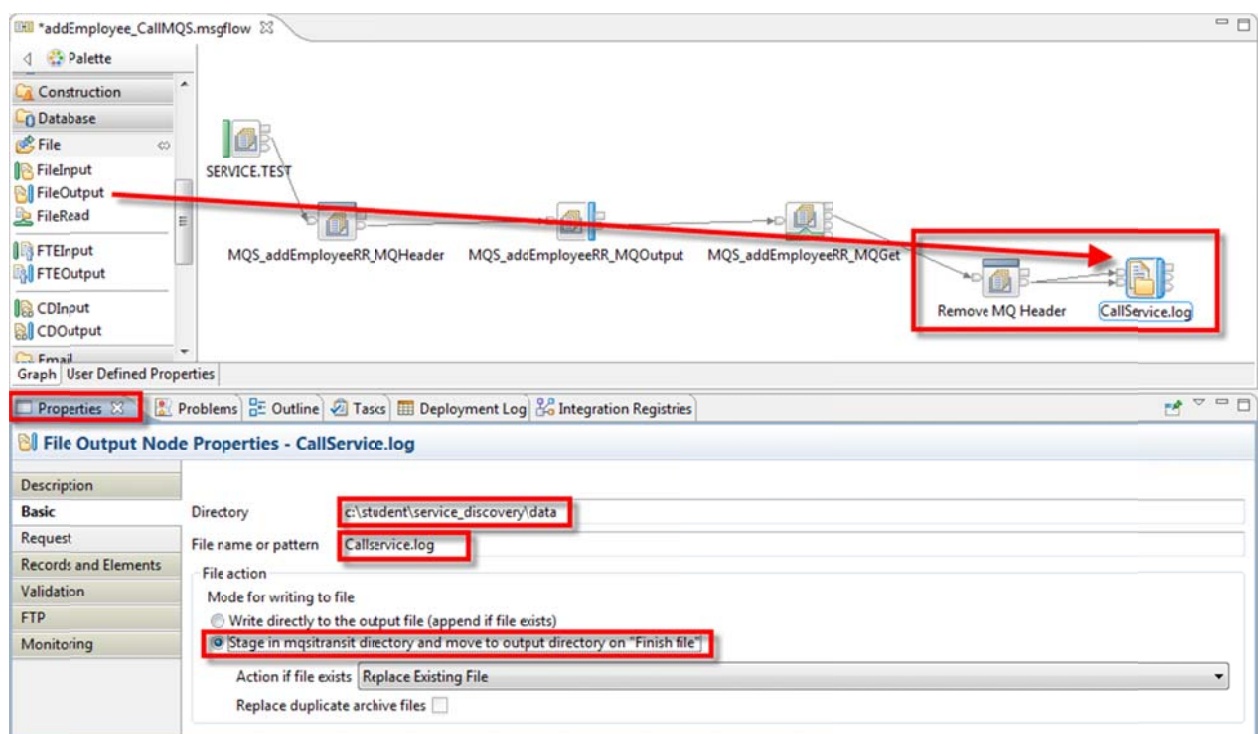
- ___11. Drag and drop a **FileOutput** node (in the **File** drawer) onto the canvas to the right of the **Remove MQ Header** node. Name the node **CallService.log**.

Wire the **Out** terminal of the **Remove MQ Header** to the **In** terminal on the **CallService.log** node. Also wire the **Out** terminal of the **Remove MQ Header** to the **Finish** terminal on the **CallService.log** node.

In the **Properties**, on the **Basic** tab, set the **Directory** path to **c:\student\service_discovery\data**

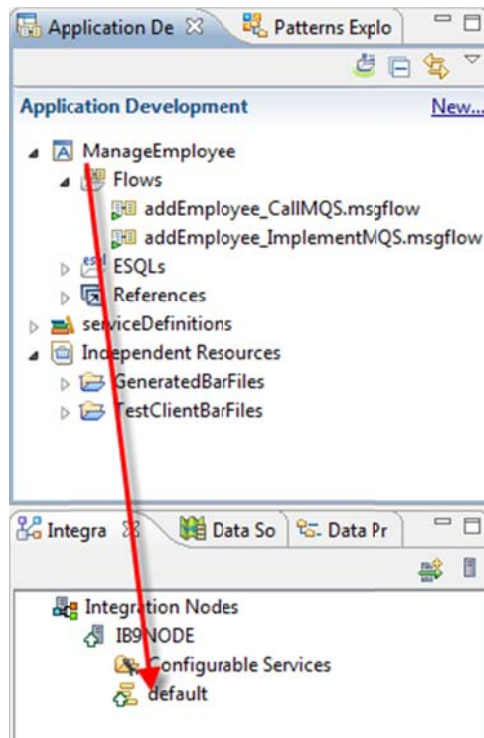
Set the **File name or pattern** to **Callservice.log**

Select **Stage in mqsitransit directory** for file action



- ___12. Save (**CTRL-S**) the message flow and close the message flow editor.

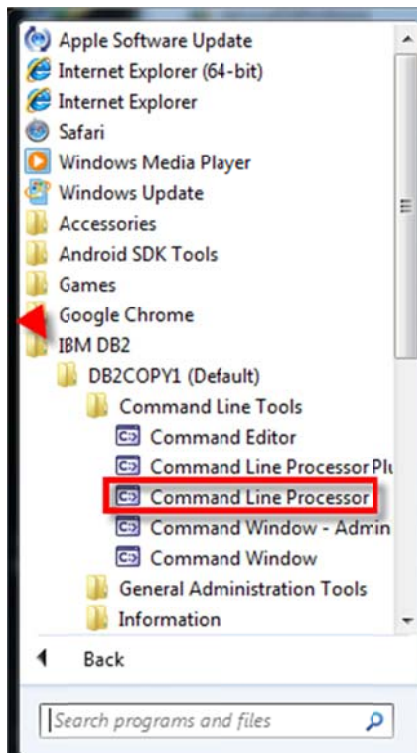
- ___13. Deploy the **ManageEmployee** Application into the **default** Integration Server (drag the Application and drop it on the Integration Server).



5.6 Test the Manage Employee application

You are now ready to test the ManageEmployee application.

- ___1. Open a DB2 command line Processor window (**Start → All Programs → IBM DB2 → DB2COPY1 → Command Line Tools → Command Line Processor**).



Enter the commands:

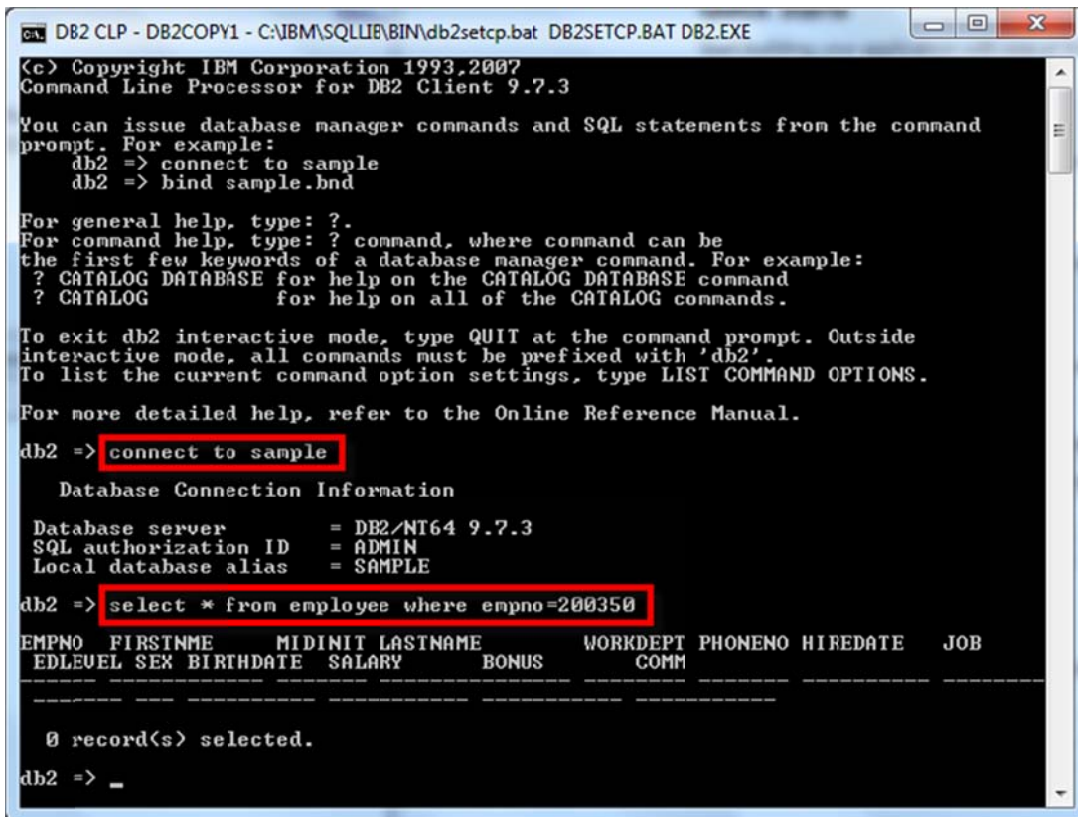
CONNECT TO SAMPLE

SELECT * FROM EMPLOYEE WHERE EMPNO=200350

If no record exists – good you will be able to add it.

If the record does exist, delete it using:

DELETE FROM EMPLOYEE WHERE EMPNO=200350



The screenshot shows a Windows command prompt window titled "DB2 CLP - DB2COPY1 - C:\IBM\SQLLIB\BIN\db2setcp.bat DB2SETCP.BAT DB2.EXE". The window contains the following text:

```
<c> Copyright IBM Corporation 1993.2007
Command Line Processor for DB2 Client 9.7.3

You can issue database manager commands and SQL statements from the command
prompt. For example:
  db2 => connect to sample
  db2 => bind sample.bnd

For general help, type: ?.
For command help, type: ? command, where command can be
the first few keywords of a database manager command. For example:
? CATALOG DATABASE for help on the CATALOG DATABASE command
? CATALOG          for help on all of the CATALOG commands.

To exit db2 interactive mode, type QUIT at the command prompt. Outside
interactive mode, all commands must be prefixed with 'db2'.
To list the current command option settings, type LIST COMMAND OPTIONS.

For more detailed help, refer to the Online Reference Manual.

db2 => connect to sample

Database Connection Information

Database server      = DB2/NT64 9.7.3
SQL authorization ID = ADMIN
Local database alias = SAMPLE

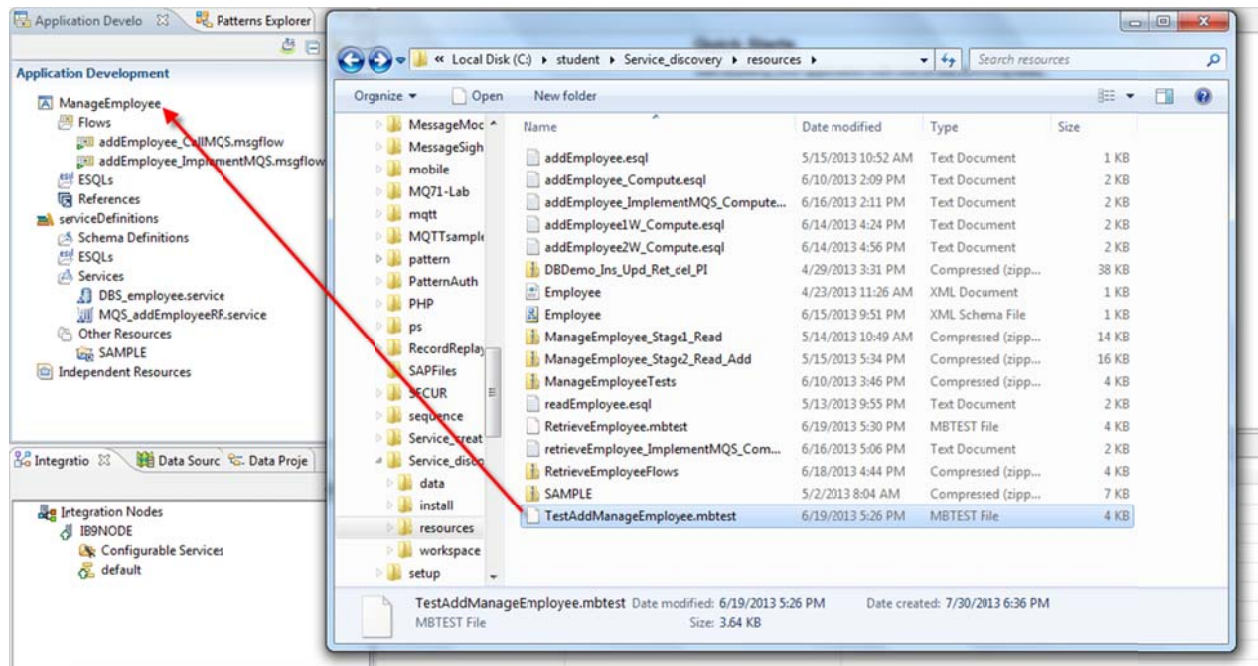
db2 => select * from employee where empno=200350

EMPNO  FIRSTME  MIDINIT LASTNAME  WORKDEPT PHONENO HIREDATE  JOB
EDLEVEL SEX BIRTHDATE  SALARY    BONUS      COMM
-----
0 record(s) selected.

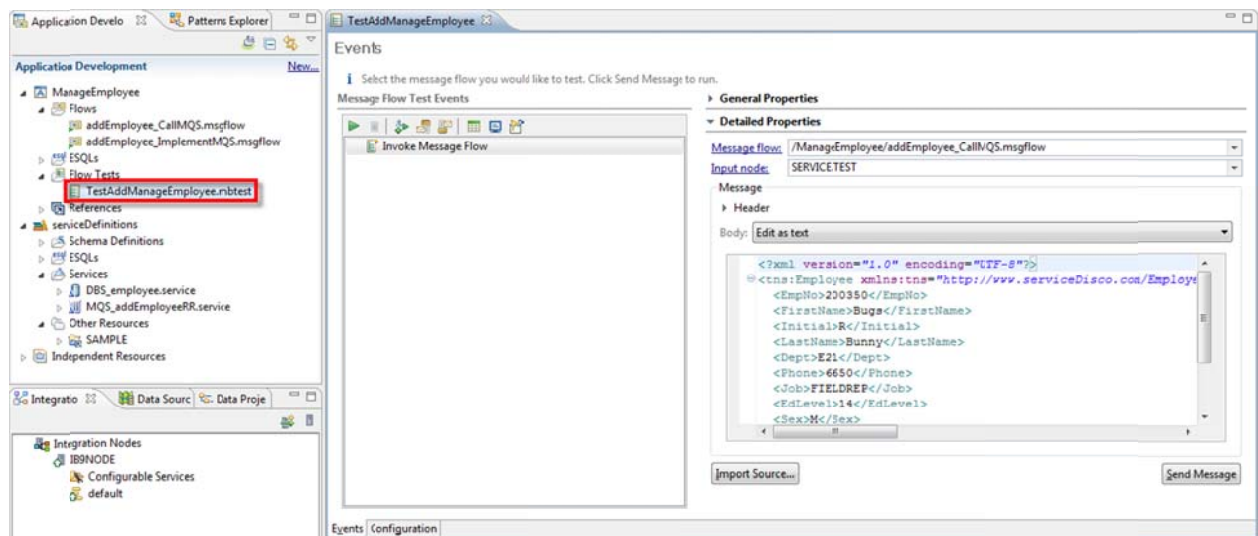
db2 => _
```

Keep this window open.

- ___2. Drag file **testAddManageEmployee.mbttest** (using Windows Explorer) from **C:\student\Service_discovery\resources** onto the **ManageEmployee** Application in the IIB Toolkit.



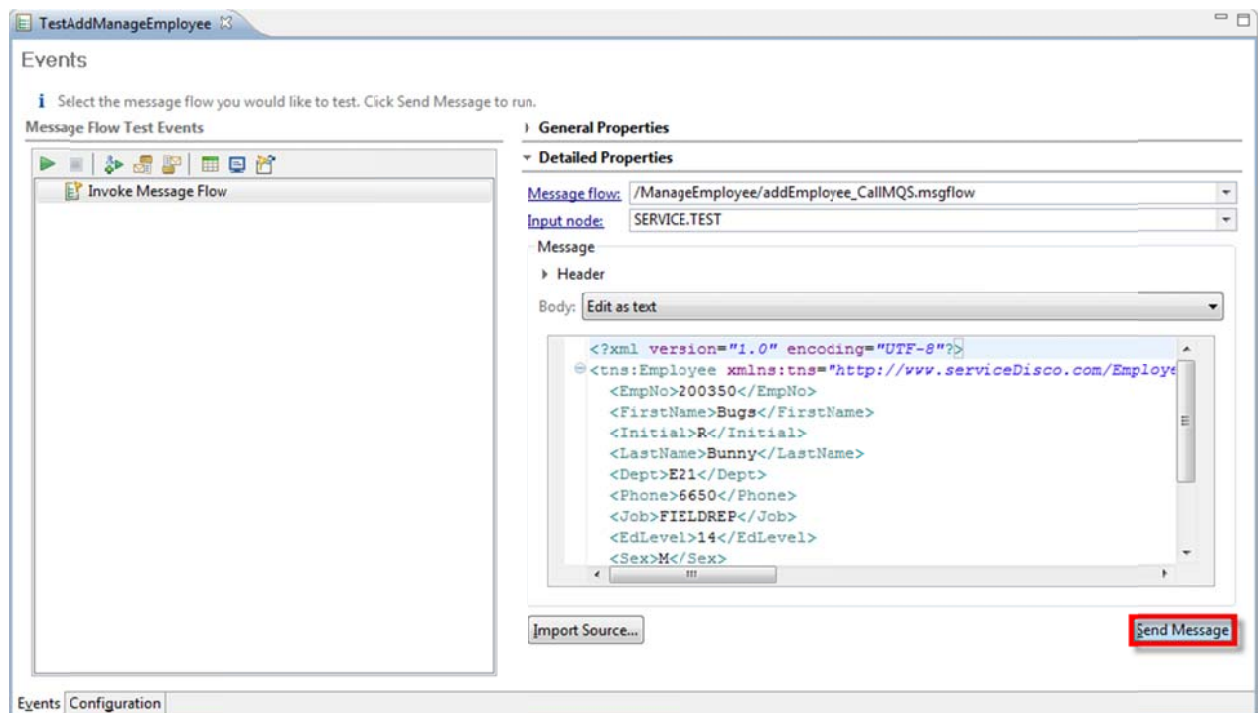
- ___3. Open the **testAddManageEmployee.mbttest** file by double clicking on it. It will be in the **ManageEmployee→Flow Tests** folder. The saved Test Client window will open.



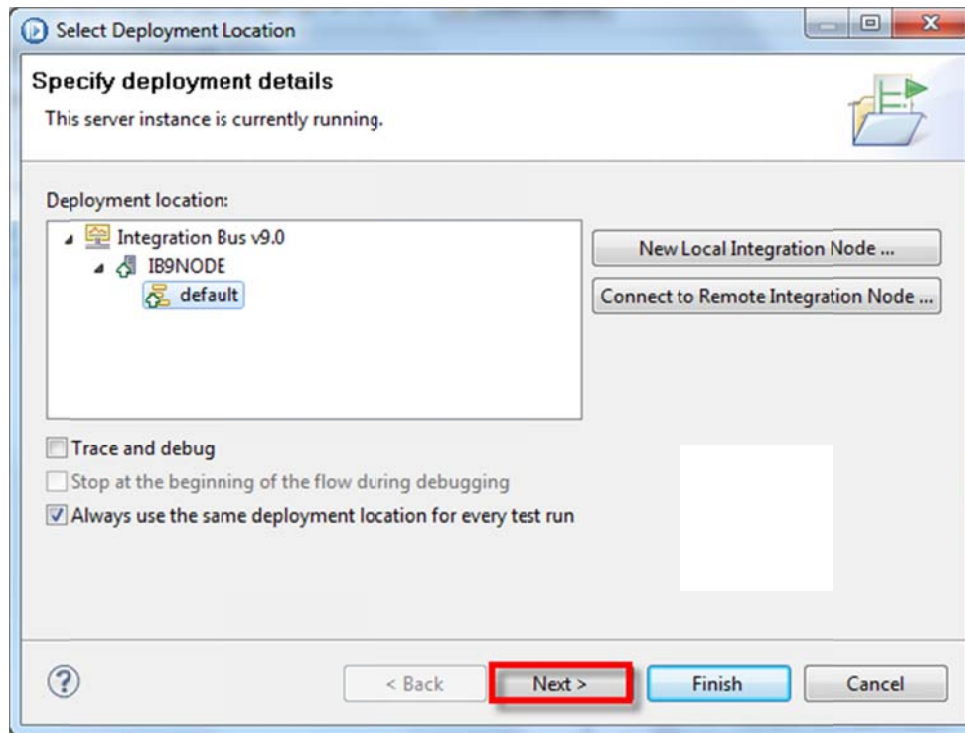
- ___4. On clicking Send Message, the XML message that is displayed will be sent to the SERVICE.TEST queue. The message will be picked up by the addEmployee_CallMQS message flow and written to the SERVICE.DISCOVERY.DB.ADD.IN queue. The message flow addEmployee_ImplementMQS will pick this request up and add the user with EMPNO=200350 to the database using the ESQL provided by implementing the Database Service DBS_employee and using the createEmployee procedure created by adding the INSERT operation in the service.

The results will be written to the file CallService.log in C:\student\service_discovery\data

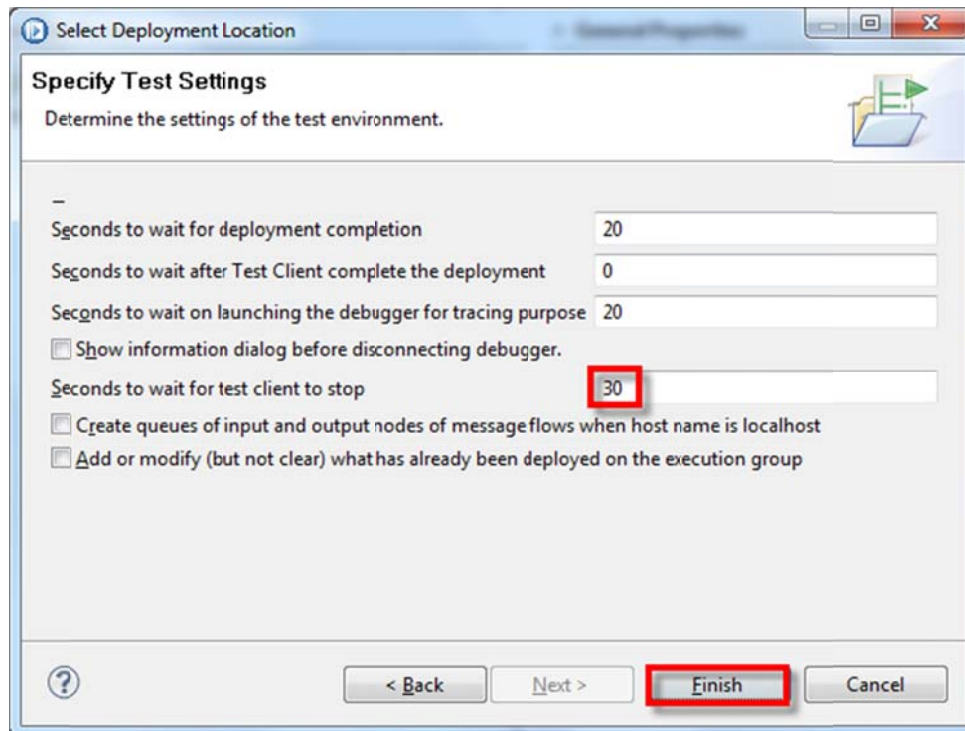
Click **Send Message**.



- __5. Select the **default** Integration Server and click **Next**.

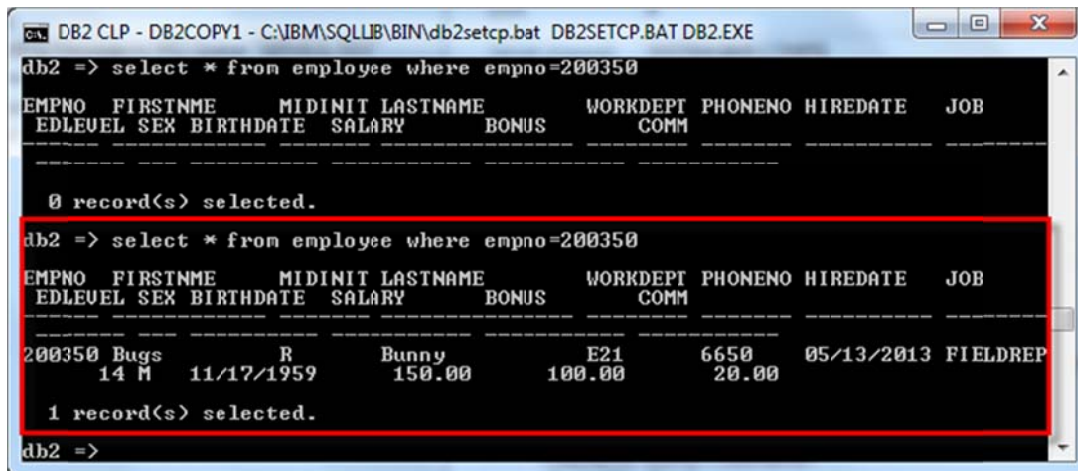


- __6. Enter **30** for **Seconds to wait for test client to stop** and click **Finish**.



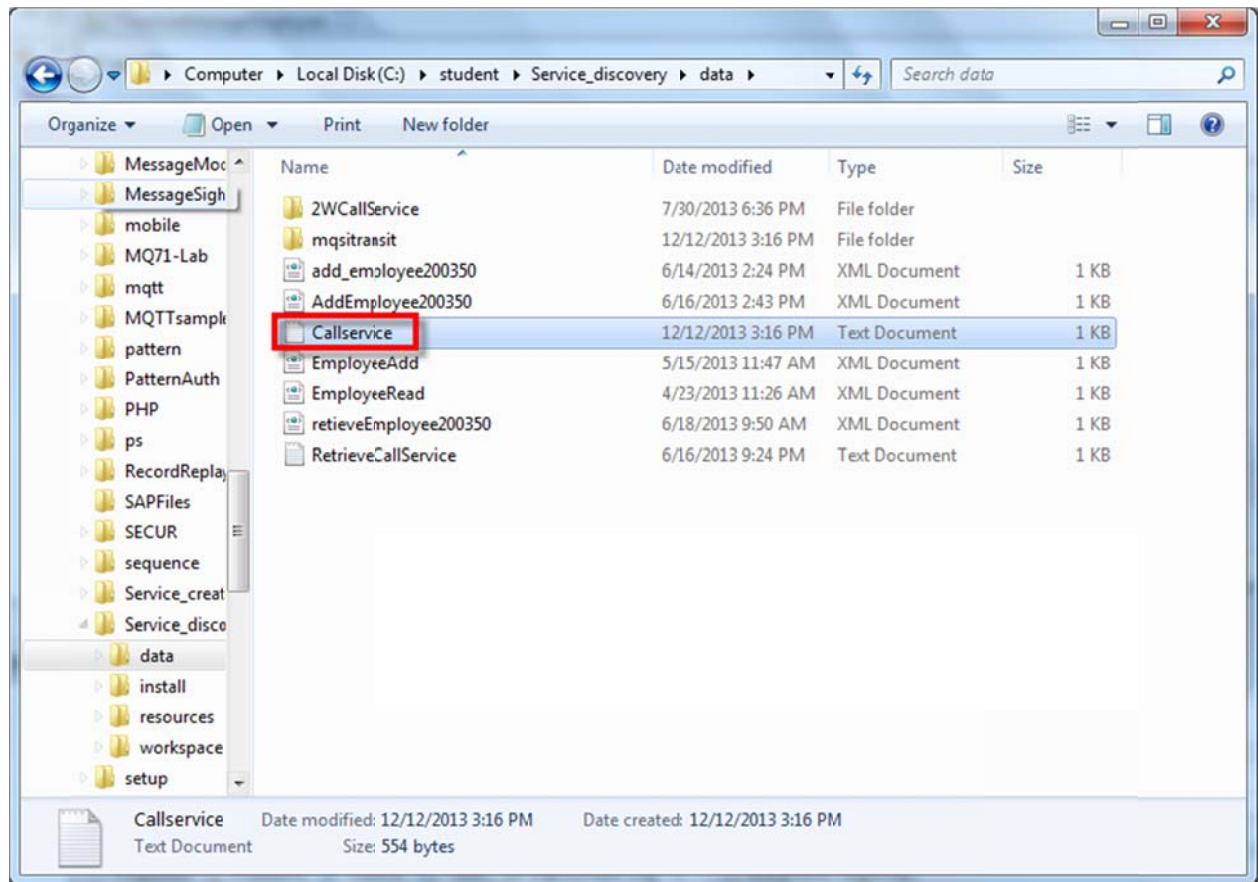
- ___7. Wait for the deployment to complete, and then the 30 seconds for the test client to stop (the output of the message flow being tested is a file, so the test client does not end automatically when the file is written; therefore, we are waiting for the test client to time out the set number of seconds).
- ___8. Go back to the DB2 Command Line Processor window and enter:

SELECT * FROM EMPLOYEE WHERE EMPNO=200350
- ___9. The select should retrieve the newly inserted record, with the EMPNO=200350.

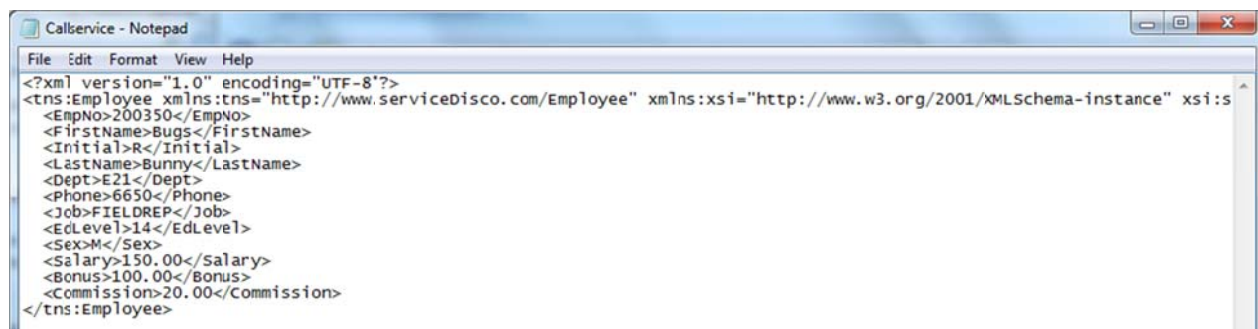


- ___10. In the DB2 Command Line Processor window, enter **quit** to exit the DB2 command line, and then enter **exit** to close the Command Line window.

- ___11. Now in Windows Explorer, move to the C:\student\Service_discovery\data directory. You should find the file named **CallService.log** newly created by the message flow.

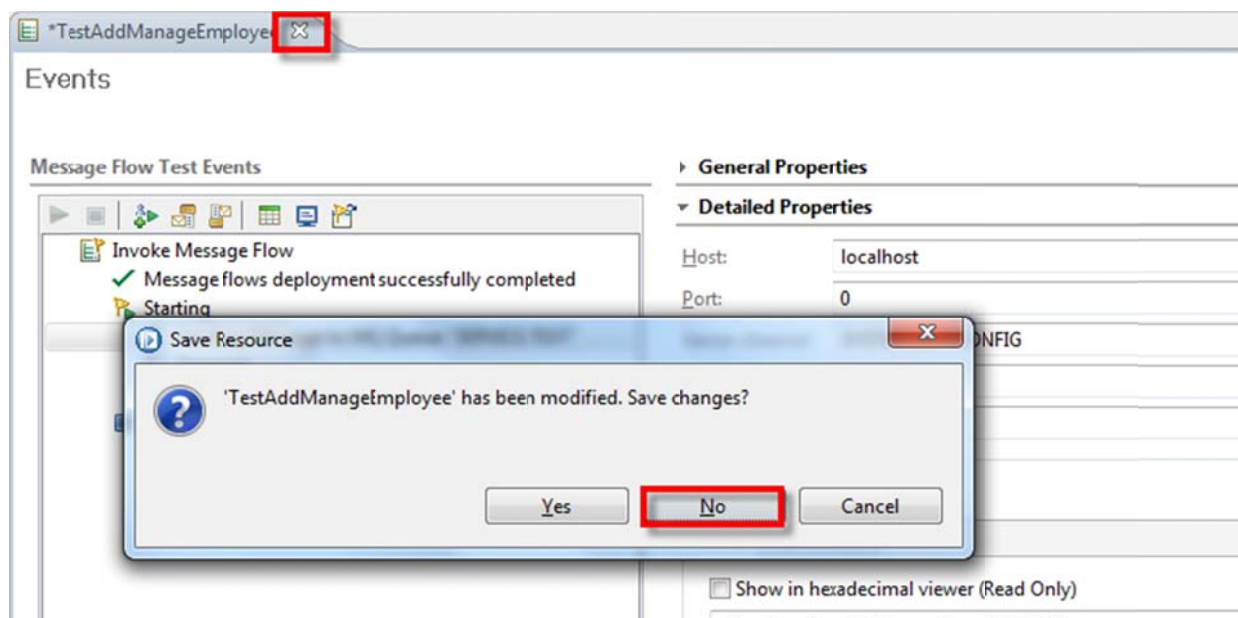


- ___12. Double-click on the file to open it.



- ___13. Close Notepad.

- ___14. Back in the IIB Toolkit, close the test client file (click on the **X**). Select **No** to not save the changes.



This concludes the Service Discovery lab.

This is the end of Lab 5.

Appendix A. Notices

This information was developed for products and services offered in the U.S.A.

IBM may not offer the products, services, or features discussed in this document in other countries. Consult your local IBM representative for information on the products and services currently available in your area. Any reference to an IBM product, program, or service is not intended to state or imply that only that IBM product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any IBM intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any non-IBM product, program, or service.

IBM may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not grant you any license to these patents. You can send license inquiries, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

For license inquiries regarding double-byte (DBCS) information, contact the IBM Intellectual Property Department in your country or send inquiries, in writing, to:

IBM World Trade Asia Corporation
Licensing
2-31 Roppongi 3-chome, Minato-ku
Tokyo 106-0032, Japan

The following paragraph does not apply to the United Kingdom or any other country where such provisions are inconsistent with local law: INTERNATIONAL BUSINESS MACHINES CORPORATION PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some states do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. IBM may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

Any references in this information to non-IBM Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this IBM product and use of those Web sites is at your own risk.

IBM may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any performance data contained herein was determined in a controlled environment. Therefore, the results obtained in other operating environments may vary significantly. Some measurements may have

been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products. Questions on the capabilities of non-IBM products should be addressed to the suppliers of those products.

All statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only.

[This information contains examples of data and reports used in daily business operations. To illustrate them as completely as possible, the examples include the names of individuals, companies, brands, and products. All of these names are fictitious and any similarity to the names and addresses used by an actual business enterprise is entirely coincidental. All references to fictitious companies or individuals are used for illustration purposes only.

COPYRIGHT LICENSE:

This information contains sample application programs in source language, which illustrate programming techniques on various operating platforms. You may copy, modify, and distribute these sample programs in any form without payment to IBM, for the purposes of developing, using, marketing or distributing application programs conforming to the application programming interface for the operating platform for which the sample programs are written. These examples have not been thoroughly tested under all conditions. IBM, therefore, cannot guarantee or imply reliability, serviceability, or function of these programs.

Appendix B. Trademarks and copyrights

The following terms are trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM	AIX	CICS	ClearCase	ClearQuest	Cloudscape
Cube Views	DB2	developerWorks	DRDA	IMS	IMS/ESA
Informix	Lotus	Lotus Workflow	MQSeries	OmniFind	
Rational	Redbooks	Red Brick	RequisitePro	System i	
<i>System z</i>	<i>Tivoli</i>	<i>WebSphere</i>	<i>Workplace</i>	<i>System p</i>	

Adobe, the Adobe logo, PostScript, and the PostScript logo are either registered trademarks or trademarks of Adobe Systems Incorporated in the United States, and/or other countries.

IT Infrastructure Library is a registered trademark of the Central Computer and Telecommunications Agency which is now part of the Office of Government Commerce.

Intel, Intel logo, Intel Inside, Intel Inside logo, Intel Centrino, Intel Centrino logo, Celeron, Intel Xeon, Intel SpeedStep, Itanium, and Pentium are trademarks or registered trademarks of Intel Corporation or its subsidiaries in the United States and other countries.

Linux is a registered trademark of Linus Torvalds in the United States, other countries, or both.

Microsoft, Windows, Windows NT, and the Windows logo are trademarks of Microsoft Corporation in the United States, other countries, or both.

ITIL is a registered trademark, and a registered community trademark of The Minister for the Cabinet Office, and is registered in the U.S. Patent and Trademark Office.

UNIX is a registered trademark of The Open Group in the United States and other countries.

Java and all Java-based trademarks and logos are trademarks or registered trademarks of Oracle and/or its affiliates.

Cell Broadband Engine is a trademark of Sony Computer Entertainment, Inc. in the United States, other countries, or both and is used under license therefrom.

Linear Tape-Open, LTO, the LTO Logo, Ultrium, and the Ultrium logo are trademarks of HP, IBM Corp. and Quantum in the U.S. and other countries.

[illegible]

NOTES



© Copyright IBM Corporation 2014.

The information contained in these materials is provided for informational purposes only, and is provided AS IS without warranty of any kind, express or implied. IBM shall not be responsible for any damages arising out of the use of, or otherwise related to, these materials. Nothing contained in these materials is intended to, nor shall have the effect of, creating any warranties or representations from IBM or its suppliers or licensors, or altering the terms and conditions of the applicable license agreement governing the use of IBM software. References in these materials to IBM products, programs, or services do not imply that they will be available in all countries in which IBM operates. This information is based on current IBM product plans and strategy, which are subject to change by IBM without notice. Product release dates and/or capabilities referenced in these materials may change at any time at IBM's sole discretion based on market opportunities or other factors, and are not intended to be a commitment to future product or feature availability in any way.

IBM, the IBM logo, ibm.com and WebSphere are trademarks of International Business Machines Corp., registered in many jurisdictions worldwide. Other product and service names might be trademarks of IBM or other companies. A current list of IBM trademarks is available on the Web at "Copyright and trademark information" at www.ibm.com/legal/copytrade.shtml.



Please Recycle
